Conception sûre et optimale de systèmes dynamiques critiques auto-adaptatifs soumis à des évènements redoutés probabilistes

Jonathan Sprauel

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Jonathan SPRAUEL

Conception sûre et optimale de systèmes dynamiques critiques auto-adaptatifs soumis à des événements redoutés probabilistes

JURY

M. Christophe BERENGUER	Membre	Professeur - Grenoble INP
M. Olivier PIETQUIN	Membre	Professeur - Université Lille 1
M. Antoine RAUZY	Rapporteur	Professeur - NTNU
M. Christian SANNINO	Encadrant	Ingénieur - Thales Avionics
Mme Christel SEGUIN	Directrice	Ingénieur de recherche - ONERA
M. Florent TEICHTEIL	Encadrant	Ingénieur de recherche - Airbus
Mme Sylvie THIEBAUX	Rapporteur	Professeur - ANU
M. Gérard VERFAILLIE	Directeur	Directeur de recherche - ONERA

École doctorale et spécialité :
EDSYS : Systèmes embarqués 4200046
Unité de Recherche :
Équipe d'accueil ISAE-ONERA CSDV
Directeur(s) de Thèse :
M. Christian SANNINO, *Mme Christel SEGUIN* et *M. Gérard VERFAILLIE*
Rapporteurs :
M. Antoine RAUZY et *Mme Sylvie THIEBAUX*

TABLE DES MATIÈRES

Conception sûre et optimale de systèmes dynamiques critiques auto-adaptatifs soumis à des événements redoutés probabilistes

Résumé

Cette étude s'inscrit dans le domaine de l'intelligence artificielle, plus précisément au croisement des deux domaines que sont la planification autonome en environnement probabiliste et la vérification formelle probabiliste. Dans ce contexte, elle pose la question de la maîtrise de la complexité face à l'intégration de nouvelles technologies dans les systèmes critiques : comment garantir que l'ajout d'une intelligence à un système, sous la forme d'une autonomie, ne se fasse pas au détriment de la sécurité ?

Pour répondre à cette problématique, cette étude a pour enjeu de développer un processus outillé, permettant de concevoir des systèmes auto-adaptatifs critiques, ce qui met en œuvre à la fois des méthodes de modélisation formelle des connaissances d'ingénierie, ainsi que des algorithmes de planification sûre et optimale des décisions du système.

Mots clés : Systèmes critiques auto-adaptatifs, Processus décisionnel markovien, Calcul de politique optimale valide, Logique Temporelle probabiliste.

Safe and optimal design of dynamical, critical self-adaptive systems subject to probabilistic undesirable events

Summary

This study takes place in the broad field of Artificial Intelligence, specifically at the intersection of two domains : Automated Planning and Formal Verification in probabilistic environment. In this context, it raises the question of the integration of new technologies in critical systems, and the complexity it entails : How to ensure that adding intelligence to a system, in the form of autonomy, is not done at the expense of safety ?

To address this issue, this study aims to develop a tool-supported process for designing critical, self-adaptive systems. Throughout this document, innovations are therefore proposed in methods of formal modeling and in algorithms for safe and optimal planning.

Keywords : Critical self-adaptive systems, Markov decision process, Computation of optimal and valid policy, Probabilistic temporal logic.

AVANT-PROPOS

À Christophe Berenguer, Olivier Pietquin, Antoine Rauzy et Sylvie Thiébaux, pour tout ce qu'ils ont dit lors de ma soutenance de thèse, et qui résonnera en moi pour des années et des années.

À Gérard, pour ses remarques toujours pertinentes.

À Florent, pour m'avoir le premier entraîné dans le monde des PCMDP.

À Christel, pour ces heures autour de Cécilia OCAS.

À Christian, pour les discussions de mécanique quantique.

À Stéphanie, pour toutes ces fois où elle nous permit d'atteindre notre optimum.

À Francine et Marc, pour leur soutien et leur aide au long des 245 pages.

À Patrick, Quentin, Fabien, Loïc et Jâcques, pour les victoires de dernières minutes.

À mes parents et mon frère, pour le fait d'être des personnes extraordinaires.

À Laurence, pour tout le reste.

Merci.

INTRODUCTION

Les appareils et systèmes conçus actuellement par l'industrie sont extraordinairement complexes et précis : un Boeing 747-400 est par exemple constitué d'environ 6 millions de pièces ; le TGV a nécessité près de 8 ans de tests et de prototypes avant sa mise en service, utilisant des technologies nouvelles et révolutionnaires ; le rover Curiosity explorant Mars depuis 2012 a été conçu pour avoir une durée de mission d'au moins 23 mois, devant faire face à des températures entre 30 et -127 degrés Celsius et à d'immenses tempêtes de sable...

De tels exemples sont nombreux et, ajoutés au fait que les systèmes informatiques embarqués dans de tels appareils deviennent de plus en plus intelligents, il semble clair qu'il devient très difficile de concevoir de tels systèmes en se basant sur une analyse manuelle. Prévoir le fonctionnement d'un système, en particulier du point de vue de la sécurité pour s'assurer que même en cas de panne aucune vie n'est mise en jeu, nécessite ainsi des outils particuliers pour modéliser des composants et des contraintes de conception, et pour prouver automatiquement que ces contraintes sont respectées.

L'émergence de nouvelles technologies implique l'émergence de nouveaux défis

En cela, le domaine de la prise de décision dans l'incertain est une branche des mathématiques appliquées en plein essor : en s'appuyant sur des techniques de **planification**, **d'optimisation sous contrainte** ou plus généralement de synthèse de politique, c'est-à-dire de stratégie, les modèles et algorithmes développés dans ce domaine permettent de planifier des décisions pour des systèmes autonomes, ou proposer des aides à la décision pour des opérateurs humains.

Cependant, la popularité nouvelle de ces techniques conduit à de nouvelles problématiques, en particulier lorsqu'elles sont appliquées dans des contextes industriels tels que les systèmes dits critiques, c'est-à-dire des systèmes dont la défaillance peut avoir des conséquences dramatiques. Pour de tels systèmes, il ne suffit plus seulement de planifier des décisions pour réagir à des évènements tels que des pannes, mais il faut aussi garantir que le système respecte des **propriétés de sécurité** : le concepteur d'un système souhaiterait par exemple concevoir des systèmes avioniques intelligents pouvant se reconfigurer en cas de panne, tout en ayant la garantie que même en prenant en compte cette "intelligence" il obtienne bien un système sûr, valide au sens du respect de certaines contraintes fixées par des autorités de certification. En résumé : peut-on créer des systèmes intelligents tout en disposant de garanties fortes sur les comportements qu'auront ces systèmes ?

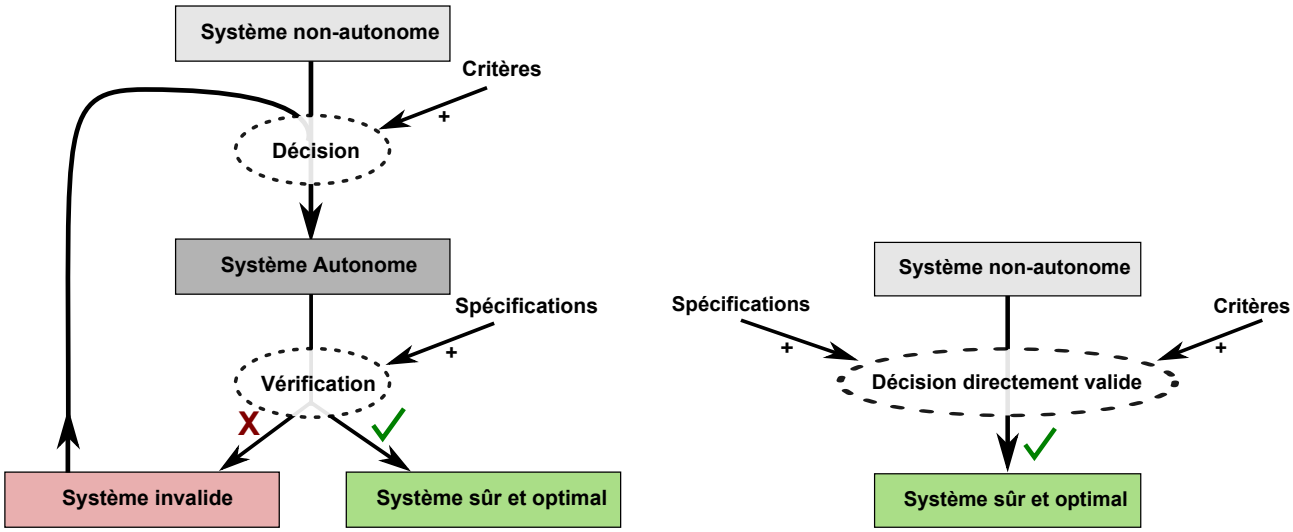
Les limites de l'approche "Décider puis Vérifier"

La solution de référence pour un tel problème est de type "Décider puis Vérifier" (Figure 1 page 2), qui consiste à appliquer de manière itérative deux résolutions : dans un premier temps résoudre le problème de décision, ce qui synthétise une stratégie de reconfiguration ; puis dans un second temps appliquer des algorithmes de validation formelle sur ces stratégies pour vérifier qu'elles sont bien valides. En revanche, si elles ne sont pas valides le concepteur du système ne peut que répéter le processus en

modifiant légèrement ses décisions, en espérant avoir cette fois une stratégie qui semble optimale et soit valide.

Il est évident que cette méthode est loin d'être idéale : d'une part la boucle "synthèse puis validation" est difficilement contrôlable puisque rien ne garantit qu'une solution serait effectivement trouvée en l'appliquant assez longtemps ; et d'autre part il n'est pas trivial de modifier les stratégies données à chaque étape d'une telle boucle, rien ne semble garantir alors que la stratégie trouvée soit la stratégie optimale.

En conséquence, les processus de conception classiques de nombreux systèmes critiques soit sont longs et coûteux, soit ne prennent pas en compte la planification des décisions. Pour de tels systèmes, il est nécessaire de trouver une autre approche, de type "Décider en vérifiant" (Figure 1 page 2).



(a) La plupart des processus actuels de conception ont un modèle "Décider puis Vérifier".

(b) Notre objectif est de construire un processus outillé autour de l'approche "Décider en vérifiant".

FIGURE 1 – Deux approches de conception de systèmes sûrs et optimaux.

Un problème de conception sûre et optimale construit sur deux domaines de recherche

Dans le cas où le système est **stochastique**, c'est-à-dire soumis à des évènements probabilistes ou possédant des variables aléatoires, le cadre le plus classique pour la planification de décision est celui des **Processus Décisionnels Markoviens** (MDP) [1], où l'on cherche à trouver les décisions à appliquer face à une situation, tout en essayant d'optimiser un gain ou un coût sur le long terme. Détaillé en particulier par Putterman [Put94], ce modèle mathématique permet de représenter un système dynamique soumis à des évènements probabilistes ; de nombreuses méthodes de résolution associées ont été développées, la plus classique étant la méthode de résolution par programmation dynamique.

Cette méthode permet de générer des politiques permettant de prendre des décisions dans l'incertain en optimisant l'espérance du coût cumulé. Du point de vue d'un ingénieur réalisant la modélisation, ce coût permet de guider la solution vers certaines voies ou d'en éviter d'autres.

Cependant, cette méthode n'est pas efficace sur le plan du temps de calcul : elle repose sur une exploration exhaustive de l'espace d'état, c'est-à-dire de toutes les configurations possibles du système, ce qui n'est pas envisageable pour des modèles de taille industrielle. De plus, le modèle MDP est limité dans son expressivité, puisque le format de définition des fonctions de coût et des fonctions de transition, représentant la dynamique de l'environnement, ne repose pas directement sur les connaissances d'ingénierie : en effet, plusieurs paramètres (au sens des variables vues par un

1. Markov Decision Process

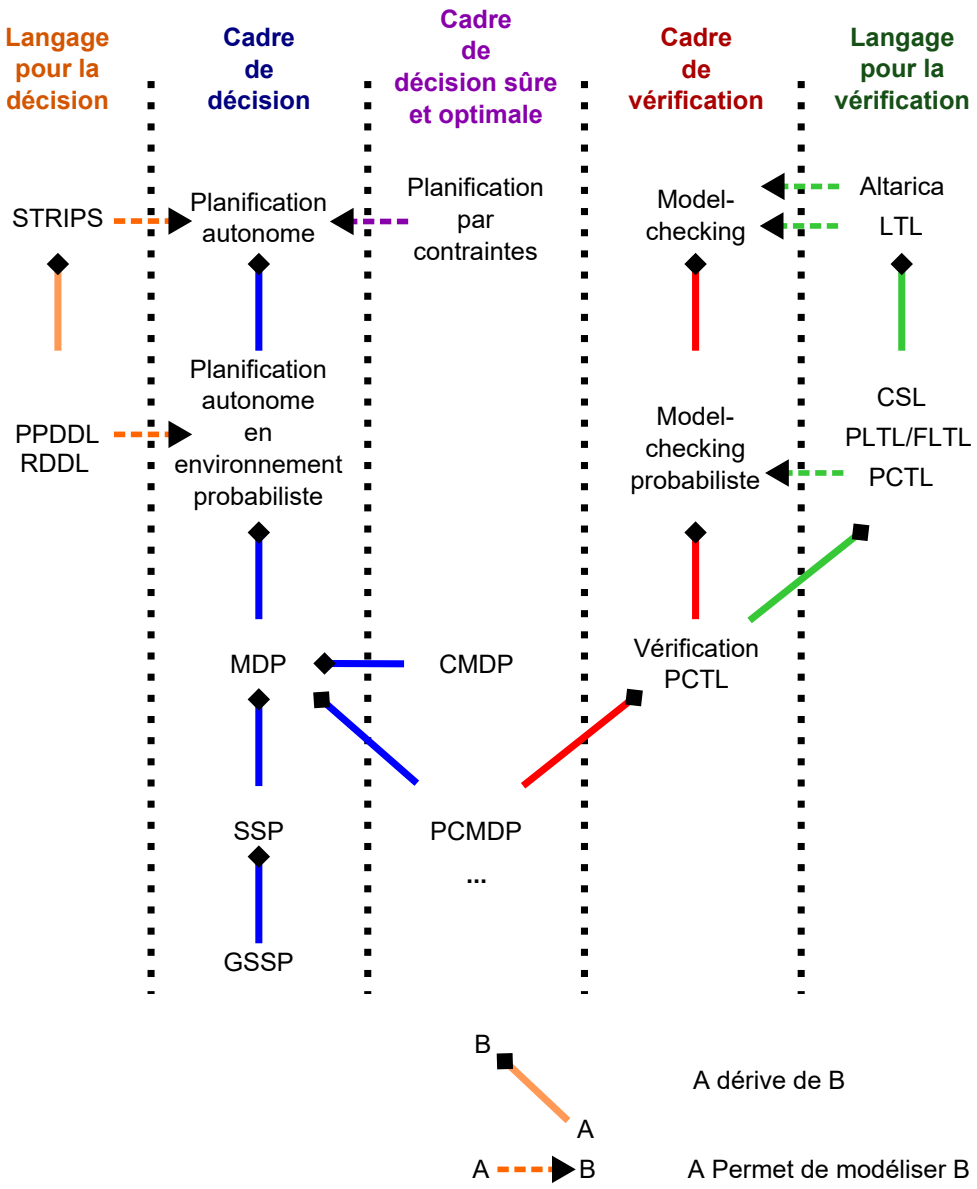


FIGURE 2 – Une courte étude bibliographique préliminaire justifie l'étude du développement d'un processus outillé de conception sûre et optimale. Plusieurs schémas de la partie État de l'art étendent et détaillent cette figure.

ingénieur du domaine tels que la position, la température,...) se retrouvent mélangés en un seul état ou un seul coût ; une étude spécifique à chaque problème est ainsi nécessaire pour faire correspondre une représentation MDP avec une représentation compréhensible où l'impact de chaque paramètre est clairement identifiable. Par ailleurs, ce modèle ne présente pas de contraintes *dures* ni de contraintes probabilistes, comme celles demandées par les autorités de certification pour les systèmes critiques. Ce modèle rentre donc dans le cadre d'une approche "Décider puis vérifier", reposant sur des tests de la stratégie trouvée pour déterminer si elle est ou non valide.

De nombreuses études ont depuis cherché à traiter chacune de ces limitations. Un des modèles notable est celui des **problèmes de plus courts chemins stochastiques** (SSP) [2] [Ber95], dont une généralisation a en particulier été étudiée par Kolobov et al. (GSSPs) [KMWG11]. Dans ce modèle, un ensemble d'états est marqué comme étant le *but* et le système doit l'atteindre avec le meilleur coût possible. De nombreuses contraintes peuvent être représentées ainsi, telles que des contraintes portant sur des états qu'il faut toujours éviter ou au contraire sur des états qu'il faut atteindre. De plus, tels qu'ont pu le montrer par exemple [HZ01] et [TKVI11], le modèle SSP a une structure particulière permettant d'obtenir des algorithmes très efficaces en termes de temps de calcul, se concentrant sur la recherche du but.

Cependant, toutes les contraintes ne peuvent pas être exprimées sous la forme d'un but à atteindre ; c'est notamment le cas des contraintes imposées par les autorités de certification qui portent sur la probabilité qu'un ou plusieurs buts ou chemins soient ou non réalisés. De plus, même lorsque les contraintes peuvent être exprimées sous la forme d'un but à atteindre, il est très difficile d'écrire directement cette transcription ; ceci est en particulier le cas lorsque la politique doit respecter plusieurs contraintes, correspondant à plusieurs buts à atteindre, qu'il faut donc exprimer sous la forme d'un seul but en ajoutant par exemple des variables de mémoire à l'espace d'état. Certaines des difficultés d'expression ont été adressées par de nombreux auteurs tels que [Alt99] ou [EKVY07], qui ont pu proposer des méthodes pour convertir d'autres formes de contraintes ou d'objectifs en un modèle SSP. Cependant, ces travaux se sont concentrés sur la gestion de ressources ou sur de la décision multi-critères, et non sur la vérification de propriétés tout au long de la vie du système, telles que par exemple le maintien de service.

Le domaine du **Model checking** fournit des outils permettant de vérifier de telles propriétés : d'une manière générale, ces méthodes formelles sont utilisées pour valider automatiquement le bon fonctionnement d'un système, qui a été modélisé dans un langage donné. Hanssen et al. en particulier [HJ94] ont spécifié un langage pour exprimer des contraintes de chemin sur des systèmes dynamiques soumis à des évènements stochastiques. Ils ont développé une méthode de calcul pour déterminer si ces contraintes sont respectées pour une chaîne de Markov donnée, c'est-à-dire pour toutes les exécutions possibles d'un système qui n'est pas contrôlable.

Ce langage, appelé PCTL pour **Probabilistic Computation Tree Logic**, a pour avantage principal d'être très expressif, puisqu'il se base sur d'autres logiques existantes (LTL, CTL [CES86]) pour les étendre avec des probabilités. Il permet en particulier d'exprimer des contraintes probabilistes sur des chemins : en effet, il est souvent impossible de s'assurer qu'un évènement critique n'arrive jamais, par exemple la perte de la capacité de pilote automatique dans un avion ; ceci est impossible parce qu'il y a toujours une faible probabilité que tous les équipements et les redondances qui servent à assurer cette capacité défaillent de façon successive. Plutôt que de tester si un tel évènement ne se produit jamais, le concepteur d'un système souhaitera alors vérifier qu'il ne se produit qu'avec une très faible probabilité. Avec le langage PCTL, il est possible d'associer une contrainte sur la probabilité qu'une situation particulière ou une succession d'évènements particuliers se produisent.

Cependant, le langage en lui-même et la méthode de calcul associée ne permettent pas seuls de trouver une stratégie satisfaisant les contraintes : il est possible de vérifier après coup si une politique donnée est valide mais, sans adaptations supplémentaires, il n'est pas trivial de créer directement la stratégie pour qu'elle soit valide. À nouveau, ces méthodes de Model-Checking probabiliste trouvent naturellement leur place dans une approche "Décider puis Vérifier".

Les travaux de Teichteil (PCMDP) [TK12a] ont pu se reposer sur ce langage pour construire un modèle mathématique, basé sur MDP et PCTL, permettant de représenter les systèmes soumis à des évènements probabilistes et les contraintes de manières unies. Puisqu'il se base sur PCTL, ce modèle prend en compte les contraintes probabilistes sur les chemins d'exécutions. Ce modèle permet une approche "Décider en vérifiant", qui garantit que les contraintes sont respectées par la politique sans nécessiter des tests supplémentaires. Cependant, les temps de calcul trop importants rendent cette méthode inutilisable sur des cas d'applications réels. De plus, le problème de capture de la dynamique

2. Stochastic Shortest Path

de l'environnement n'est pas résolu : les modèles PCMDP sont tout aussi complexes à écrire que les modèles MDP.

Plusieurs langages de modélisation se sont attaqués au problème de la capture d'un modèle stochastique, parmi lesquels on peut citer les travaux de Younes [YMS03] sur le langage PPDDL : il s'agit d'un langage permettant de capturer des modèles MDP au format STRIPS, qui est un format utilisé par de nombreux solveurs de planification. Dans les format STRIPS et dans PPDDL, un problème est décrit par des variables abstraites appelées fluents, ainsi que des actions sur ces variables abstraites, ce qui permet de s'affranchir de la définition explicite des états et transitions. Cependant, ce langage est centré uniquement sur la décision, il ne permet pas par exemple d'exprimer des contraintes de type PCTL. À l'inverse, d'autres langages de modélisation se concentrent sur l'expression et la vérification de contraintes, tels que par exemple le langage AltaRica développé notamment par Arnold et al. [APGR99] ; mais les langages provenant de la communauté du Model checking ne permettent pas nativement d'exprimer des problèmes de décision. Pour que ces langages puissent être utilisés pour exprimer des modèles du type de PCMDP, il serait ainsi nécessaire de les modifier pour y ajouter soit l'expression de contraintes PCTL, soit la prise en compte d'actions contrôlables.

Les études réalisées précédemment montrent donc qu'un large panel de modèles est disponible sur le plan mathématique (Figure 2 page 3), permettant de capturer des systèmes dynamiques soumis à des évènements probabilistes puis de synthétiser des stratégies optimales ou d'en vérifier la validité au regard des contraintes. Ces modèles permettent de traiter toute la gradation des besoins, entre décision (MDP, GSSP), respect de contraintes (PCTL) et combinaison des deux aspects selon une approche "Décider en vérifiant" (CMDP, PCMDP).

Cependant, aucune des méthodes actuelles ne permet d'être appliquée raisonnablement dans un cadre industriel : les modèles les plus simples ne présentent pas une expressivité suffisante pour prendre en compte la dynamique du système et les contraintes de Model-Checking en même temps. À l'inverse, les modèles les plus complexes ne proposent pas d'algorithme de résolution suffisamment efficace sur le plan du temps de calcul. Enfin, la capture du modèle depuis des connaissances d'ingénierie est imparfaite, puisque les méthodes actuelles de capture du modèle sont détachées d'un contexte industriel.

La démarche de développement d'un processus outillé de conception sûre et optimale

C'est ce qui justifie cette étude. Cette étude consiste à développer un processus outillé, permettant de concevoir des systèmes auto-adaptatifs critiques, qui intègre comme composante une méthode de modélisation formelle des connaissances d'ingénierie, dans le but d'obtenir une planification sûre et optimale des décisions du système lorsqu'il doit réagir à des évènements probabilistes.

Ce problème de planification sûre et optimale consiste en pratique à développer une approche de **synthèse de contrôleurs en environnement probabiliste directement valides**, qui ne nécessitent pas d'être validés par la suite par d'autres outils. Deux questions très générales motivent donc cette étude :

- Comment prendre une décision sur un système critique, c'est-à-dire comment éliminer ou privilégier certains choix en fonction de contraintes sur l'exécution future ?
- Comment prendre en compte le futur dans la décision, c'est-à-dire comment prendre en compte des évènements potentiels (probabilistes) tout en se limitant aux futurs qui semblent les plus pertinents ?

Ces deux questions illustrent immédiatement le problème fondamental auquel nous nous confrontons : celui d'une alliance à trouver entre le respect des contraintes et l'optimisation ; entre les garanties mathématiques de sécurité et la recherche de la meilleure décision possible sur le long terme.

À la suite d'un état de l'art étendu et de définitions essentielles, la démarche exposée **dans le chapitre III** a consisté dans un premier temps à déterminer quel modèle de la littérature serait le

mieux à même de représenter le problème de conception sûre et optimale, si un tel modèle existe, ou d'être en mesure de proposer un nouveau modèle s'il n'en existe pas. Pour cela un cas d'application industriel a été construit et étudié à partir d'un scénario informel avionique, portant sur l'aide à la décision pour la maintenance d'un business jet. Puis une définition mathématique de ce scénario a été formalisée, afin de déterminer quelles hypothèses sont ou non raisonnables dans un contexte industriel. Enfin, les modèles existants de la littérature ont été évalués en regard de ces hypothèses. Cette modélisation a été publiée dans un article à la conférence Lambda-mu-19 [SSS14].

L'étape suivante a consisté **dans le chapitre IV** à obtenir une version simplifiée du modèle PCMDP, appelée SPC MDP, en sélectionnant des hypothèses supplémentaires en cohérence avec le cas d'application industriel, ce qui implique de développer un algorithme de résolution et d'en prouver la validité. Pour cela, un modèle basé sur PCMDP a été défini, qui se limite à des contraintes déterministes. Un algorithme de résolution a alors pu être implémenté, en prouvant qu'il retourne une solution valide et optimale. Enfin, les performances de cet algorithme ont été évaluées, en termes de temps de calcul, sur un ensemble de cas de tests académiques. Ce modèle a été publié dans un article à la conférence AAAI-14 [SKTK14].

Dans le chapitre V, l'objectif a alors été de valider le modèle SPC MDP sur le cas industriel. La première étape de cette validation a été de concevoir un outil de saisie des données par l'utilisateur, afin de générer un modèle formel du problème du business jet. La seconde étape a alors été de récupérer automatiquement les informations manquantes depuis des documents d'ingénierie. Enfin, ce processus complet a été évalué sur plusieurs instances du problème, notamment sur le plan du temps de calcul et de la qualité du plan de réparation obtenu.

Dans le chapitre VI, cette même démarche est appliquée sur un autre cas d'application, mettant en valeur de manière plus explicite les avantages d'un processus outillé de conception sûre et optimale. Pour cela, des scénarios de décision dans le contexte avionique ont été étudiés, afin de sélectionner un cas industriel d'aide à la décision sûre et optimale pour lequel les données d'entrées peuvent être modélisées formellement. Ce cas d'application a alors été exprimé sous une forme mathématique nouvelle, en définissant un modèle de gestion de modes dégradés, basé sur une formalisation de la qualité de service. Enfin, ce cadre mathématique a été comparé avec les modèles existants dans la littérature.

Dans le chapitre VII, l'enjeu a alors été de revenir sur le modèle PCMDP afin de développer un algorithme de résolution permettant de traiter des problèmes de grande taille, comme le problème de gestion de modes dégradés. Des concepts classiques, utilisés pour améliorer les performances des algorithmes de recherche dans la littérature, ont été adaptés à notre cadre particulier. Puis une formalisation de l'algorithme a été réalisée, ce qui implique de fournir des preuves de terminaison, d'optimalité et de complétude. Enfin, les performances de cet algorithme ont été évaluées sur un ensemble de cas de tests académiques.

La dernière étape a consisté **dans le chapitre VIII** à évaluer le processus outillé sur la conception d'un système industriel concret. Ceci a été réalisé au travers d'une capture sous forme de modèle d'un sous-ensemble d'une architecture avionique, représentant une chaîne de données utilisée par la fonction RNP-AR, correspondant à une fonction permettant à l'avion d'effectuer une approche avec une très haute précision. Ce cas d'application industriel a dans un second temps été exploité pour l'obtention de logiques d'alerte et d'évaluation de la conséquence d'une défaillance, en particulier à des fins de validation amont. Enfin, les performances et l'ergonomie du processus outillé ont été évaluées au regard de ce retour d'expérience.

CHAPITRE I

CONTEXTE ET FONDEMENTS

Sommaire

DANS un premier temps, il nous faut préciser le cadre de cette étude. La caractérisation de ce cadre passe par la définition du vocabulaire et des concepts mathématiques que nous considérerons comme fondamentaux - nous les considérerons comme admis et étudiés - ainsi que par le rattachement de notre étude aux différents domaines mathématiques existants.

Comme nous le verrons, les questions abordées dans cette étude sont généralisables à plusieurs domaines applicatifs. Nous ne présenterons donc pas le contexte industriel dans lequel nous avons réalisé ces travaux - par exemple nous ne ferons pas référence à des projets industriels ou des équipements en développement actuellement dans l'industrie ; en revanche, nous ferons à plusieurs reprises dans les chapitres suivants une description des besoins industriels existants, lorsque notre réflexion nous amènera à nous poser des questions sur ce sujet. Nous pouvons de plus préciser que ces travaux ont été initiés par une collaboration entre **l'ONERA** et **Thales Avionics**, ce qui est la raison du choix des différents cas d'applications que nous traiterons dans cette étude.

I.1 Éléments de notations

Avant d'aborder des aspects plus théoriques, nous détaillons ici les termes, abréviations et symboles usuels que nous utiliserons dans cette étude. La plupart des abréviations seront détaillées et expliquées dans des chapitres ultérieurs, mais cette section pourra néanmoins servir de référence lexicale lorsqu'une de ces abréviations aura été utilisée dans un autre chapitre que celui où elle aura été définie.

I.1.1 Termes usuels

Tout au long de cette étude, nous noterons en **gras** les termes communément admis dans les communautés Planification, Model-Checking, Diagnostic et Avionique, ou les termes admis directement depuis une citation. Ces termes représentent des concepts considérés connus par tous, dont nous rappelons ou précisons simplement le sens au regard de notre problème.

Nous appellerons **définition** la description d'un concept formel qui sera admis sans démonstration. Ce concept servira de point de départ à des preuves ; nous considérerons comme admis toutes les hypothèses et démonstrations amont à cette définition. Par extension, nous placerons dans des encadrés "définition" certains concepts communément admis mais souvent non formalisés, pour lesquels nous proposons une formalisation adaptée au cadre de cette étude.

Définition 1 (*Exemple de définition*)

Une définition.

Nous appellerons **théorème** un résultat que nous prouverons à partir des définitions et théorèmes établis. Il s'agit d'un résultat considéré comme principal.

Théorème 1

Un théorème.

Nous appellerons **lemme** un résultat mineur que nous prouverons à partir des définitions et théorèmes établis. Il s'agit d'un résultat mineur, ou d'un résultat intermédiaire permettant de prouver par la suite un théorème.

Lemme 1

Un lemme.

Nous appellerons **propriété** un résultat corollaire digne d'intérêt, mais non utilisé par un résultat principal. Il s'agit d'une remarque pertinente mais qui ne sera pas utilisée par la suite.

Propriété 1

Une propriété.

Nous appellerons **algorithme** un extrait de langage informatique ou une procédure décrite en pseudo-code. Cette procédure présente des données d'entrée et une sortie, et décrit la succession d'opérations permettant de calculer la sortie à partir de l'entrée. Le pseudo-code utilisé est similaire aux langages BASIC, C/ C++ ou Java.

Algorithm 1: Un exemple d'algorithme

```
1 Un algorithme.
```

Enfin nous encadrerons les résultats que nous avons obtenus à partir d'un retour d'expérience. Il s'agit de résultats tels que des concepts de formalisation ou des protocoles de tests, que nous avons appliqués dans notre étude ; bien que nous expliquerons la démarche nous ayant menés à ce résultat, le fait de les marquer sous la forme d'un encadrement - et non d'un théorème - montre que d'autres méthodes auraient pu être envisagées.

Un exemple d'encadrement

Un encadrement.

I.1.2 Abréviations usuelles

et al.	et alii : et collaborateurs
i.e.	id est : c'est-à-dire
LTL	Linear Temporal Logic : Logique Temporelle Linéaire [Pnu77]
MDP	Markov Decision Process : Processus Décisionnels Markoviens [Put94]
MEL	Minimum Equipment List [EASb]
PCMDP	Path Constrained Markov Decision Process : MDP à chemin contraint [TK12a]
PCTL	Probabilistic Computation Tree Logic : Logique Temporelle probabiliste par arbre [HJ94]

I.1.3 Récapitulatif des symboles

& ou $\wedge$	et
|| ou $\vee$	ou
! ou $\neg$	non
$\forall$	pour tout
$\exists$	il existe
$\in$	dans, appartient à
∞	infini
π	une politique
X	un ensemble ou une variable aléatoire
x	un élément d'un ensemble
$P(x\|y)$	la probabilité de x sachant y
$\#X$ ou $\|X\|$	le nombre d'éléments dans X (cardinal)
$x \in [0, 1)$	x peut prendre les valeurs de 0 inclus à 1 non-inclus

I.2 Systèmes critiques

Comme nous l'avons évoqué en introduction, le contexte de notre étude est celui des **systèmes critiques**. Deux concepts très vastes sont désignés derrière ces deux mots : le terme de "système" est utilisé dans des expressions très différentes telles que système informatique, système social, système industriel, ou encore système digestif. La définition que nous choisissons est celle de [Vil88] :

Définition 2 (*Système*)

Un système est un ensemble déterminé d'éléments discrets interconnectés ou en interaction.

I.2.1 Contextes sur les systèmes dans le domaine avionique

Cette définition (Définition 2 page 10) est compatible avec les définitions du domaine telles que [ALRL04] : un système est une entité interagissant avec d'autres entités ; ces entités peuvent être des objets physiques tels que des ordinateurs, des logiciels, des opérateurs humains ou encore le monde physique composé de phénomènes naturels. Cette définition est cependant trop générale pour notre étude ; nous choisissons donc de restreindre le type de systèmes considérés à celui du domaine avionique. Nous donnons de ce domaine la définition suivante (Définition 3 page 10), basée sur la description de [Hel10] :

Définition 3 (*Avionique*)

L'avionique est l'ensemble des équipements électroniques, électriques et informatiques qui aident au pilotage des aéronefs et des astronefs dans l'espace aérien ou extraplanétaire.

Parmi les types de systèmes particuliers qui sont rencontrés dans le domaine avionique, on parle de **systèmes embarqués** pour désigner des systèmes qui sont complètement contenus dans un dispositif, tels que les différents calculateurs à bord d'un avion ; l'une des particularités de ce type de système est qu'ils doivent souvent respecter des exigences très particulières, par exemple sur le fait que les calculs doivent être effectués en temps réel. Néanmoins, les équipements embarqués à bord d'un avion interagissent aussi avec des équipements au sol, par exemple pour la gestion du trafic aérien, donc un certain nombre d'études sur les systèmes avioniques comportent à la fois des composants embarqués et des composants extérieurs.

Bien que nous nous concentrerons particulièrement sur les systèmes qui font partie du monde avionique, les méthodes que nous étudierons sont applicables sur l'ensemble des systèmes dits **complexes** : ce terme désigne des systèmes ayant de nombreux composants interagissant à plusieurs niveaux, à plusieurs échelles ou dans plusieurs disciplines, ce qui les rend difficiles à comprendre ; le terme "complexe" ne se réfère cependant pas exclusivement à la quantité de composants dans le système, puisque certains systèmes peuvent composer de nombreux composants mais être très répétitifs donc simple à comprendre. Nous choisissons la définition suivante (Définition 4 page 10), issue de l'[ARP 4754] (Aerospace Recommanded Practice) mais que nous reprenons de [Ade11] :

Définition 4 (*Complexité*)

La complexité est un attribut d'une fonction, d'un système ou d'un composant donnant à leurs opérations, à leurs modes de défaillance où à l'effet de leurs défaillances un caractère difficile à appréhender sans l'aide de méthodes analytiques.

Nous choisissons de restreindre notre étude aux systèmes complexes puisque cette complexité justifie le fait qu'il soit nécessaire d'utiliser des techniques de modélisation pour comprendre le système. D'une façon générale, nous pouvons donner la définition suivante de la modélisation (Définition 5 page 11), issue de [Die00] :

Définition 5 (*Modélisation*)

*Un **modèle** est un système de symboles permettant de rendre compte de la plupart des perceptions dont on dispose lorsque nous souhaitons décrire un phénomène (observé ou imaginé) afin de l'interpréter intelligiblement.*

*La **modélisation** est l'opération par laquelle on établit un modèle d'un phénomène, afin d'en proposer une représentation interprétable, reproductible et simulable.*

Nos travaux se placent donc dans le contexte de la modélisation des systèmes critiques, plus précisément dans le cadre de la modélisation des systèmes dynamiques critiques ayant une structure statique : il s'agit de systèmes dont la structure - c'est-à-dire l'ensemble des composants du système et des liens entre ces composants - est fixe mais dont le comportement change au cours du temps. Nous pouvons choisir la définition suivante (Définition 6 page 11), inspirée de [HK02] :

Définition 6 (*Système dynamique*)

Un système dynamique est un système dont le comportement évolue dans le temps, habituellement selon des règles inchangées.

Un avion complet est bien évidemment un système dynamique, puisque son comportement change au cours du temps ; le système électrique de l'avion est aussi un système dynamique, puisque les différents niveaux d'énergie sont amenés à changer au cours du temps, mais le plus souvent la structure du réseau électrique est statique, c'est-à-dire qu'aucun câble électrique n'est débranché ou rebranché entre les éléments du réseau. C'est dans ce sens que nous considérons des systèmes dynamiques, mais à structure statique.

Un type particulier de systèmes dynamique est celui des systèmes autonomes, c'est à dire des systèmes tels que des robots, des drones ou des programmes informatiques suffisamment intelligents pour pouvoir remplir leurs fonctions sans l'intervention d'un être humain, ou en collaboration à niveau égal avec un être humain. Ici, le terme "autonome" ne qualifie pas réellement le niveau d'intelligence du système, on effectue plutôt une distinction entre le point de vue du système et le point de vue de l'observateur extérieur. On prendra la définition suivante (Définition 7 page 11), venant de [Mül02] :

Définition 7 (*Système autonome*)

Un système autonome est une entité capable d'organiser ses activités pour réaliser des tâches déterminées sans intervention humaine.

Il existe plusieurs manières de formaliser l'autonomie d'un système, comme le présentent les travaux de Muller [Mül02], mais le point central est qu'un système autonome est composé d'une logique interne réagissant à des entrées et des sorties fournies par l'environnement ; du point de vue de l'observateur extérieur, seul le comportement du système est observable, la logique interne n'est pas accessible et ne peut pas être maitrisée directement. Cette logique interne peut correspondre à un ensemble de règles, par exemple "si j'ai un obstacle en face de moi, je fais une rotation à gauche", ou le plus souvent à la solution d'une équation différentielle dont certains paramètres varient avec le temps, par exemple une équation ajustant l'accélération du moteur pour maintenir une voiture à une certaine vitesse constante. D'une manière générale, on parle de **contrôle** [LM67] pour désigner la règle mathématique ou logique qui permet au système d'avoir le bon comportement en fonction des entrées qu'il perçoit depuis l'environnement ; on parlera de **synthèse de contrôleur** pour désigner le fait de concevoir ces règles de comportement. Cependant, ces termes sont imprécis en dehors d'un contexte applicatif et recouvrent des méthodes et des objets très différents ; nous les définirons avec plus de rigueurs par la suite.

Si l'environnement extérieur est vu comme une perturbation, par exemple des conditions météorologiques qui viendraient diminuer la capacité d'un drone autonome à communiquer avec le sol, alors on parle le plus souvent dans la littérature de système auto-adaptatif pour désigner le fait que le comportement du système s'adapte à ce changement et cherche à conserver un certain niveau de performance. Nous pouvons choisir la définition suivante (Définition 8 page 12), extraite de [Cha08] :

Définition 8 (*Système auto-adaptatif*)

Un système « auto-adaptatif » est un système capable de modifier sa configuration ou sa structure interne pour prendre en compte les changements de son environnement et ainsi offrir un service d'une qualité optimale.

L'un des mots clés importants de cette définition est "optimale" : en effet, il est nécessaire de spécifier en quoi le comportement du système est mieux ou moins bien. Le terme désigné dans la littérature est celui de **critère**, pour désigner un ensemble de paramètres que l'on souhaite maximiser ou minimiser. Par exemple, un drone peut chercher à toujours avoir le meilleur signal radio - son critère est de maximiser la force du signal - et ses actions tendront à remplir cet objectif ; parfois, plusieurs critères peuvent cohabiter et il est nécessaire de trouver une forme d'équilibre entre ces différents objectifs : c'est par exemple le cas pour le guidage GPS d'une voiture, qui chercherait à la fois à minimiser le temps de trajet mais aussi à réduire le coût financier en évitant au maximum les péages. On parlera de problème **multi-critères** pour désigner ce type de problèmes.

Par conséquent, nous pouvons situer notre problème dans un premier domaine : nous considérons des systèmes auto-adaptatifs, c'est-à-dire des systèmes ayant des objectifs définis sous la forme de critères, devant réaliser ces objectifs de façon autonome, et pouvant pour cela adapter leurs comportements (dynamiques) dans le temps ; pour concevoir le comportement (le contrôleur) d'un tel système, nous pouvons réaliser une modélisation afin de pallier sa complexité ; enfin, nous privilégierons dans nos études des systèmes de ce type qui peuvent être rencontrés dans le domaine avionique.

I.2.2 Contexte sur la criticité

Le second concept, celui de "critique" désigne les conséquences d'un dysfonctionnement du système : nous reprenons ici la définition de [Keh05], qui qualifie un système de **critique** lorsque son dysfonctionnement a un impact important sur son environnement matériel, humain ou économique (Définition 9 page 12).

Définition 9 (*Système critique*)

Un système critique est un système dont le dysfonctionnement a un impact dramatique, allant de pertes coûteuses jusqu'à des risques pour des humains.

La notion d'impact doit bien évidemment être définie pour chaque système : imaginons un scénario où un robot tombe dans un trou ; la première question que l'on peut se poser est "s'agit-il de quelque chose de grave ou non ?". La réponse à cette question dépend du contexte, puisqu'elle sera probablement "non" s'il s'agit d'un robot jouet et "oui" s'il s'agit d'un robot martien. Si la réponse est "non", il n'est pas nécessaire d'étudier plus en avant ce dysfonctionnement ; si la réponse est "oui", alors on considère que ce dysfonctionnement est un **évènement redouté** qui nécessite une étude particulière. Dans le cas du robot, on peut imaginer plusieurs solutions pour pallier cet évènement, la plus évidente étant de boucher le trou ; si cela n'est pas possible, comme dans le cas d'un robot martien, on peut par exemple envisager de placer plusieurs capteurs sur le robot pour faire en sorte qu'il ne tombe pas dans le trou.

Cependant, un nouveau problème se pose puisque les capteurs peuvent tomber en panne ; il subsiste donc un risque que l'évènement redouté ne survienne, c'est à dire que le robot tombe dans le trou. La seconde question qui se pose alors est "dans quelles conditions cet évènement redouté est acceptable ?". Dans le domaine avionique, la réponse à cette question est apportée en premier lieu par des textes de loi ([CS-25]) qui sont repris par des décrets applicatifs faisant office de standards ([ARP 4754],[ARP 4761]) qui décrivent les phases de certifications, en particulier en fixant des objectifs de sécurité que le système et les sous-systèmes doivent atteindre. On parle plus précisément **d'analyse fonctionnelle des dangers** (FHA) pour désigner l'analyse consistant à identifier les conditions de défaillance affectant la sécurité de l'avion et à les classifier en fonction de la gravité de leurs effets (Tableau 7 page 205). Ainsi, un dysfonctionnement du système qui n'entraînera qu'une simple augmentation de la charge de travail de l'équipage sera classé dans la catégorie des défaillances "mineures", tandis qu'un dysfonctionnement du système entraînant plusieurs décès ou blessures graves sera classé "catastrophique". Il est important ici de définir un concept, celui de défaillance, qui désigne le fait qu'un système ne peut plus remplir

sa fonction ; nous prenons la définition suivante (Définition 10 page 13), inspirée du standard [ARP 4754] :

Définition 10 (*Défaillance*)

*Une **défaillance** désigne le fait qu'un élément ou un ensemble d'éléments ne sont plus aptes à accomplir leur fonction.*

À partir de la catégorisation des défaillances, le standard [ARP 4754] (basé sur les textes de lois [CS-25]) définit des objectifs de sécurité sous la forme d'un certain nombre de défaillances de sous-composants ou d'une certaine probabilité qu'une défaillance survienne par heure de vol ; plus la gravité des effets d'une défaillance est élevée, plus la probabilité de défaillance doit être faible et le nombre de défaillances de sous-composants doit être grand. Dans le cas de notre robot par exemple, si on juge que l'évènement redouté "tomber dans un trou" a des conséquences catastrophiques, alors le système doit être conçu avec suffisamment de redondances ou des équipements suffisamment fiables pour garantir que les capteurs ne peuvent tomber en panne que très rarement - avec une probabilité inférieure à 10^{-9} par heure.

Pour effectuer le lien avec la définition de Système, nous définissons notre contexte comme celui des systèmes à évènements discrets : il s'agit simplement de systèmes dont les composants peuvent subir des modifications instantanées de la part de l'environnement, telles que des défaillances ; le terme "discret" est à opposer à continu : dans un système à évènements continus, un évènement survient de façon continue, bien que l'observateur puisse choisir un certain pas de temps pour observer cet évènement. C'est par exemple le cas pour de l'eau qu'on réchauffe : l'eau chauffe en permanence, de façon continue ; à l'inverse, un évènement discret peut par exemple correspondre à l'arrivée d'un nouveau client à un guichet automatique, ou le passage d'une vitesse dans une voiture. Notons que ceci ne présage en rien de la dynamique du système, qui peut être continue : par exemple, le passage d'une vitesse dans une voiture est un évènement discret, mais la régulation de la consommation de carburant de la voiture qui suit ce passage de vitesse est une dynamique continue. Pour des systèmes à dynamique continue et évènements discrets, on parle dans la littérature de **systèmes hybrides** ; ce type de système est cependant à la limite des systèmes que nous considérons dans notre étude - la plupart des systèmes avioniques que nous considérerons seront modélisés sous l'angle d'une dynamique discrète. Pour la définition d'un système à évènements discrets (Définition 11 page 13), nous prenons la définition suivante, inspirée de [Atl12] :

Définition 11 (*Système à évènements discrets*)

Un système à évènements discrets est un système pour lequel le comportement est un ensemble de transitions possibles entre différents états suite à l'occurrence d'évènements ponctuels sur le plan temporel.

Notons que le concept d'évènements discrets est plus général que celui des systèmes critiques ; cependant, nous l'introduisons à cette étape de notre réflexion pour mettre en avant le fait que ce concept est commun à la fois aux communautés s'intéressant aux systèmes auto-adaptatifs et aux communautés s'intéressant aux systèmes critiques.

En termes de modélisation, un système à évènements discrets correspond le plus souvent à une vision macroscopique, permettant de décrire la chronologie des actions et des évènements des sous-systèmes plutôt que de chercher à unifier tous les modèles de comportement de ces constituants. Ce type de systèmes peut être représenté par de nombreuses méthodes, en particulier sous la forme d'automates ou de réseaux de Petri. Le choix de la méthode adaptée dépend de certaines hypothèses sur le comportement du système et sur les évènements. En particulier, les cas avioniques présentent souvent plusieurs évènements possibles, chacun pouvant potentiellement survenir à n'importe quel instant ; ce type de situation correspond à un des modèles dits "**non-déterministes**" (avec modèle de temps arborescent). On peut par exemple imaginer un robot martien pouvant subir plusieurs pannes de capteurs, chacune de ces pannes pouvant survenir à n'importe quel moment ; l'enjeu lors de la conception du robot est alors de prévoir les bons équipements, la bonne structure ainsi que les règles de comportement (contrôles) appropriées pour que le robot puisse continuer sa mission malgré les pannes pouvant survenir.

Si on dispose d'une information supplémentaire sur les évènements, sous la forme d'une probabilité de défaillance dans un faible pas de temps, alors on parlera **d'évènements probabilistes**. Ces informations sont utiles pour évaluer le risque de perte d'une fonction particulière, par exemple pour savoir que l'un des capteurs du robot est plus susceptible de tomber en panne qu'un autre ; c'est ce type d'évènements que l'on rencontre en majorité dans les systèmes avioniques puisque, comme nous l'avons évoqué précédemment, les standards avioniques imposent une étude des probabilités de défaillance (Tableau 6 page 204).

I.2.3 Contexte sur les exigences

À partir des définitions précédentes, nous pouvons donc situer notre domaine comme celui des systèmes auto-adaptatifs critiques soumis à des évènements probabilistes, c'est-à-dire des systèmes auto-adaptatifs subissant certains évènements ponctuels qui peuvent l'amener dans des situations redoutées. Le fait que le système soit auto-adaptatif et critique rend sa conception difficile, en particulier sur le plan de la synthèse de contrôleur (les logiques de comportement), puisqu'il faut faire en sorte que le système optimise un critère tout en respectant certaines contraintes de sécurité.

Nous pouvons souligner la différence entre *critère* et *contrainte* : un critère est un objectif que le système cherche à optimiser, tandis qu'une contrainte est un ensemble de règles que le système cherche à satisfaire. Dans les deux cas, les critères et contraintes sont exprimés en fonction de la dynamique du système.

Exprimer des critères et des contraintes est une tâche complexe, qui peut être réalisée de multiples manières comme nous le verrons dans l'état de l'art. La méthode d'expression appropriée dépend de certaines caractéristiques du problème ; par exemple, en termes de critère à optimiser on peut faire une distinction entre les **critères additifs**, qui additionnent une certaine valeur dans le temps, ou encore les **critères moyens** qui effectuent la moyenne de ces valeurs dans le temps ; le choix d'un critère ou d'un autre dépend de ce que l'on cherche à obtenir comme comportement : un critère moyen sera par exemple adapté si l'on parle d'un drone devant maintenir un signal à un certain niveau constant, tandis qu'un critère additif sera adapté si l'on s'intéresse à minimiser une distance de trajet entre plusieurs points de passages.

Cette multiplicité des modèles et méthodes est aussi vraie pour les contraintes de sécurité ; on peut en particulier distinguer les **contraintes sur des états**, par exemple le fait qu'une voiture a le droit ou non de tourner à gauche à une certaine intersection, et les **contraintes sur les chemins**, par exemple pour exprimer le fait qu'un drone ne doit jamais aller au-dessus d'une certaine altitude, quels que soient les évènements pouvant survenir ou les actions choisies par le drone. La nuance ici est que les contraintes sur les chemins sont plus générales, puisqu'elles concernent plusieurs états successifs et expriment des conditions vis à vis de séquences d'évènements ou d'actions, tandis que les contraintes sur des états sont indépendantes du comportement suivi par le système dans le passé ou du comportement qu'il suivra : une fois qu'une voiture a passé une intersection avec une contrainte sur le fait de tourner à gauche, il peut oublier cette contrainte, sauf s'il revient à la même intersection ; en revanche, le drone doit à tout instant choisir des actions qui ne l'amèneront pas au-dessus d'une altitude donnée, malgré les défaillances pouvant survenir.

Il est possible d'exprimer de nombreux autres types de contraintes, par exemple des contraintes sur un critère (additif ou moyen) tel que la consommation totale de carburant lors d'un trajet entre plusieurs villes. Dans le contexte qui nous intéresse, c'est-à-dire celui des systèmes avioniques, on s'intéresse le plus souvent aux trois contraintes (sur des chemins) suivantes :

- **Sûreté logique** : quelque chose de mauvais ne se produit jamais.
- **Vivacité** : quelque chose de bien continue de se produire (infiniment souvent).
- **Atteignabilité** : il existe un moyen pour que quelque chose se produise.

Des définitions plus précises, en termes de Logique Temporelle, sont disponibles dans les travaux de [Pnu77], mais nous reviendrons aussi sur ces définitions par la suite.

Ces deux définitions générales de critères et de contraintes nous permettent de mieux définir notre problème : nous souhaitons concevoir les règles de comportement d'un système auto-adaptatif de façon à ce qu'il puisse optimiser certains critères tout en respectant des contraintes, dans un environnement

où certains évènements probabilistes ponctuels peuvent survenir. Pour désigner ce type de problème, nous parlerons dans la suite de **conception sûre et optimale**.

I.2.4 Contexte sur les processus

Pour réaliser une conception sûre et optimale, plusieurs méthodes existent actuellement dans l'industrie. La méthode la plus classique est de réaliser une modélisation de la dynamique du système, des exigences (contraintes) et des critères, puis d'effectuer un processus permettant de vérifier si les exigences sont respectées. Dans le cadre des systèmes en général, on parle alors de **Vérification et Validation** pour désigner ce processus ; en s'inspirant de la définition de l'[ARP 4754] : la validation correspond à la vérification que les spécifications sur un produit sont suffisamment complètes et correctes, tandis que la vérification correspond aux tests permettant de dire que la réalisation du système est correcte, c'est à dire que toutes les spécifications applicables ont été mises en place de façon correcte dans le système.

Dans le domaine avionique, les processus de Vérification et Validation sont détaillés dans des standards tels que le standard [ARP 4754] - qui décrit le processus global de développement d'un avion - tout en étant appuyés par des standards tels que l'[ARP 4761] décrivant les méthodes possibles pour réaliser des analyses de sécurité sur le système, ou le standard [DO 178C] décrivant les méthodes spécifiques au développement logiciel. Nous ne donnerons pas de détail sur ces standards, auxquels un lecteur intéressé pourra se référer, puisque nous considérons des cas d'applications qui sont plus généraux que ceux décrits dans ces standards, par exemple des cas opérationnels : si on se place du point de vue d'une compagnie aérienne, souhaitant optimiser la rentabilité d'un avion tout en respectant des exigences de sécurité imposées par des autorités de régulation, alors certains des concepts que nous avons définis sont toujours applicables (système critique, auto-adaptatif, critère, contrainte,...) mais les méthodes pour mettre en œuvre ces concepts sont différentes des méthodes qui sont détaillées dans les standards avioniques. La similarité entre certains des concepts opérationnels et les concepts d'ingénierie système est cependant suffisamment grande pour qu'il soit intéressant de ne pas restreindre notre réflexion à l'un ou l'autre des domaines.

Cependant, l'exemple des standards avioniques nous permet de mettre en avant le fait que le respect des standards est souvent acquis au travers (1) d'un processus de modélisation, par exemple au travers de modèles tels que les arbres de défaillance ou les chaînes de Markov dans le domaine de la sécurité avionique, et (2) d'outils d'aide à la conception qui assistent certaines des tâches ; on pense notamment aux outils Catia de Dassault Systems [Sys], Scade de Esterel Technologies [Tec] ou encore Capella de PolarSys [Pol]. Nous pouvons retranscrire ceci au travers de deux définitions (Définition 12 page 15) : la définition de **processus**, issue de l'[ARP 4754], qui exprime le fait qu'un système est obtenu suite à une succession d'actions de la part de l'utilisateur, puis la définition de processus outillé qui précise que ce processus peut être porté par des outils spécifiques à chacune des tâches du processus.

Définition 12 (*Processus outillé*)

Un processus est un ensemble d'activités reliées les unes aux autres, dont l'application permet de produire une sortie ou un produit donné.

Un processus outillé fait référence à un processus dont la mise en œuvre s'appuie sur des outils qui ont été adaptés spécifiquement à ce processus.

Un processus outillé prend souvent la forme d'une succession d'outils informatiques permettant de saisir, modifier, transformer ou vérifier la connaissance d'un système. Puisque le processus est composé d'étapes successives, chacune des étapes peut être amenée à produire des documents ou des résultats, qui peuvent être utilisés pour valider le processus complet, par exemple pour obtenir une validation du système auprès d'autorités de certification. À nouveau, il est possible de renvoyer un lecteur intéressé aux standards [ARP 4754] pour de tels exemples de résultats intermédiaires.

Enfin, nous pouvons exploiter l'ensemble des considérations de contexte que nous avons explicitées précédemment pour définir notre problématique ; nous formulerons en première approche la notion de "processus outillé de conception sûre et optimale" de la manière suivante (Définition 13 page 16) :

Définition 13 (*Processus outillé de conception sûre et optimale*)

Un processus outillé de conception sûre et optimale d'un système critique est un processus outillé permettant de développer un système auto-adaptatif critique, c'est-à-dire répondant à :

- *des contraintes de sécurité,*
- *et des critères de qualité du service rendu.*

Notons que l'approche "Décider puis Vérifier" est alors un des moyens possible de mettre en œuvre la conception sûre et optimale, avec les défauts que nous avons évoqués en introduction.

I.3 Fondements mathématiques

En termes de placement vis-à-vis de la littérature existante, ces travaux s'appuient sur deux domaines mathématiques : la planification dans l'incertain d'une part, cherchant à trouver les meilleures décisions répondant à un critère précis, et d'un autre côté les Logiques Temporelles qui permettent d'exprimer des contraintes sur l'évolution d'un système, et donc sur ces mêmes décisions.

Une analogie simple qu'il est possible de prendre est celui d'un jeu d'échec : le but d'un joueur est de trouver une succession de décisions le faisant gagner face à un adversaire imprévisible. La dimension de *prévision dans l'incertain*, vient ici de la prise en compte des coups possibles de l'adversaire dans sa décision : le joueur cherche un plan conditionnel, comme par exemple "s'il joue la reine, je jouerais la tour ; puis s'il joue son pion je jouerais mon fou...".

Une contrainte en Logique Temporelle sur un jeu d'échec pourrait être "je ne veux jamais être en échec" : le joueur souhaite, en prenant sa décision, que cette condition soit toujours respectée quelle que soit l'évolution possible du système, c'est-à-dire quels que soient les coups joués par l'adversaire.

Trouver une solution optimale sur ce même jeu pourrait être "je veux gagner en un minimum de coups" : le joueur dispose d'un critère à optimiser, par exemple un critère additif tel que le nombre de mouvements, et il recherche le meilleur plan possible pour le maximiser ou le minimiser. De façon similaire, pour le concepteur d'un système critique, un bon système est un système respectant une contrainte forte sur toutes les évolutions possibles du système ("ne jamais être dans une situation dramatique") et un paramètre à optimiser ("trouver les actions les plus efficaces pour mener à bien la mission").

Dans cette section, nous détaillons ces deux domaines afin de faciliter la compréhension de certaines considérations que nous aurons par la suite. Cette section détaille donc les définitions usuelles et explicite les notations qui seront utilisées tout au long des chapitres suivants.

I.3.1 Chaînes de Markov

Propriété de Markov

L'outil mathématique de référence pour raisonner sur l'évolution d'un système en environnement incertain est celui des Chaînes de Markov [Nor98]. Le principe de base est de considérer qu'à chaque instant le système est dans un état x. Dans le cas général, cet état peut être décrit par un ensemble de variables discrètes ou continues, et pour les systèmes que nous considérons toutes ces variables sont supposées *observables* (il est possible de connaître leur valeur précise à chaque instant).

Ce système va évoluer, passant de l'état x à un autre état x'. Cette évolution n'est pas déterministe : à partir de l'état initial x, le système a une certaine probabilité d'arriver dans l'état x'_1, une probabilité d'arriver dans x'_2, ... Il n'est plus possible de dire précisément quels sont les états successifs du système, et on définit alors l'état du système au temps t comme la variable aléatoire X_t.

La notion de **variable aléatoire** signifie le plus souvent que X est une grandeur numérique dépendant des résultats d'une expérience aléatoire : par exemple, si on effectue plusieurs jets de dés, alors on peut calculer pour chaque jet la somme des valeurs des dés ; ceci définit une variable aléatoire, comme étant la somme de la valeur des dés, pour laquelle il est possible de définir une loi de probabilité : pour deux dés à 6 faces, le chiffre 7 est la valeur la plus probable tandis que les chiffres 2 et 12 sont les moins probables. Dans notre cas, nous prenons une définition générale de la variable aléatoire : au lieu d'être une fonction à valeur dans l'espace des nombres réels, il s'agit d'une fonction à valeur dans l'espace des états, et qui dépend potentiellement de tous les jets aléatoires qui ont eu lieu précédemment.

Pour revenir à notre variable aléatoire X_t, on parlera de temps continu si t prend une valeur réelle positive, et de temps discret lorsque t prend ses valeurs dans un ensemble discret $(t_0, \ldots, t_n, \ldots)$. Si on considère un temps discret et un espace d'états discrets (c'est-à-dire que toutes les variables considérées sont discrètes), on obtient donc un arbre dont les nœuds sont les états successifs possibles et les branches sont les transitions entre ces états, étiquetées par une probabilité de transition. Cet arbre, de profondeur infinie, est appelé **chaîne de Markov** lorsque ces probabilités de transitions respectent une certaine propriété : elles ne doivent dépendre que de l'état courant, et non de l'exécution passée du système.

Définition 14 (*Propriété de Markov (temps discret)*)

Soient $(X_k)_{(0\leq k\leq n)}$ *variables aléatoires sur un espace E,*
Soit $(i_0, \ldots, i_{n-1}, i, j) \in E^{n+2}$ *une suite donnée d'éléments de E,*

La variable aléatoire X_{n+1} *respecte la propriété de Markov lorsque :*

$$\Pr(X_{n+1} = j | X_0 = i_0, X_1 = i_1, \ldots, X_{n-1} = i_{n-1}, X_n = i) = \Pr(X_{n+1} = j | X_n = i)$$

(X_k) *est une chaîne de Markov homogène lorsque :*

$$\forall n \geq 0, \forall (i_0, \ldots, i_{n-1}, i, j) \in E^{n+2},$$
$$\Pr(X_{n+1} = j | X_0 = i_0, X_1 = i_1, \ldots, X_{n-1} = i_{n-1}, X_n = i) = \Pr(X_1 = j | X_0 = i)$$

Détails : La propriété de Markov signifie, de manière classique, que le système est "sans mémoire" : il ne dépend du passé que par l'état courant. L'hypothèse d'une chaîne de Markov homogène se traduit par le fait que la dynamique de l'environnement est supposée indépendante du temps : une situation donnée mènera toujours au même ensemble de conséquences et avec les mêmes probabilités. Les systèmes que nous souhaitons modéliser respecteront vraisemblablement cette hypothèse, que nous considérerons donc acquise dans la suite.

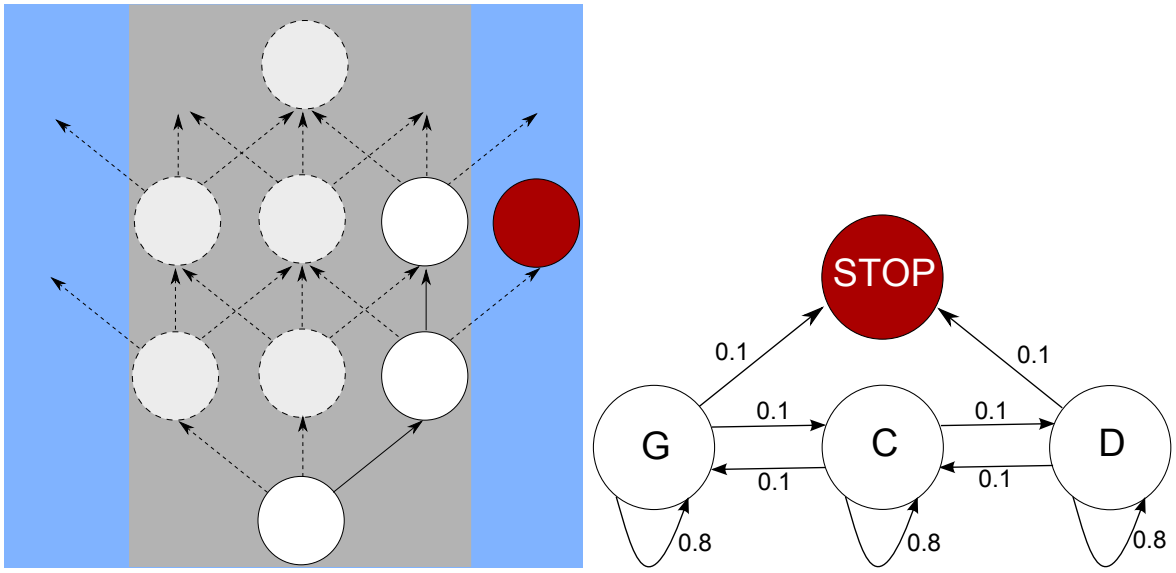
Exemple académique : marche aléatoire

L'un des apports principaux d'une chaîne de Markov est de pouvoir obtenir une représentation compacte de l'arbre des évolutions possibles du système : les paragraphes précédents décrivent un arbre de profondeur infini, dont les chemins correspondent à toutes les évolutions possibles du système ; dans cet arbre, la propriété de Markov permet de contracter toutes les occurrences de chaque état, c'est-à-dire qu'un état x_0 sera toujours le même qu'il apparaisse au sommet de l'arbre ou après une dizaine de transitions. Cette contraction, décrite dans la figure (3) nous permet de concevoir des méthodes pour prédire et contraindre l'évolution du système. Cette figure décrit l'exemple classique d'une personne avançant sur un pont ; à chaque étape, la personne avance et a une chance (0.1) de dévier sur la droite ou une chance (0.1) de dévier sur la gauche. L'arbre de gauche décrit donc les évolutions possibles du système, chaque état étant un cercle et chaque transition une flèche. Si on simule l'évolution du système pas à pas, on obtient une *trace*, par exemple "centre, droite, droite" sur la figure.

La figure de droite montre la version contractée du système : il y a seulement 4 états possibles, avec des transitions passant de l'un à l'autre associées à une certaine probabilité.

Dans les cas d'applications considérés par ces travaux, une chaîne de Markov représente ainsi l'évolution d'un système lorsqu'il n'est pas contrôlé : un état est décrit par l'ensemble des systèmes fonctionnant ou non, et une transition est généralement l'arrivée d'une défaillance, avec une certaine probabilité connue. L'hypothèse de Markov se traduit alors par le fait que la loi de défaillance, c'est-à-dire la probabilité qu'une panne survienne durant une certaine durée, ne dépend ni de l'historique du système ni du temps écoulé depuis la mise en service du système : il s'agit d'une loi exponentielle, décrite à partir d'un **taux de défaillance** noté généralement λ.

Dans un tel modèle, centré sur les défaillances pouvant survenir, les probabilités de transitions de la chaîne de Markov ne dépendent pas du temps, mais seulement des autres pannes pouvant se produire : en effet, à partir d'un état donné, le concepteur d'un système cherche simplement à déterminer quelle est la prochaine défaillance pouvant survenir, et non précisément dans combien de temps elle va se produire. La représentation en chaîne de Markov permet donc ici de s'abstraire complètement du temps, ou plus précisément de ne considérer que la notion de séquence et d'oublier le temps précis qui s'écoule entre deux changements d'états.



(a) Arbre représentant toutes les évolutions possibles d'un système

(b) Graphe compact de la chaîne de Markov correspondante

FIGURE 3 – Une chaîne de Markov

Notons que, connaissant la loi de chaque défaillance pouvant survenir, il est possible de connaître exactement le temps moyen avant la prochaine défaillance ; notons aussi que certains modèles, dits à **temps continu**, permettent dans une certaine mesure de conserver la notion de temps, ce qui est par exemple nécessaire dans des modèles où toutes les lois ne sont pas exponentielles.

I.3.2 Validation formelle

Logiques Temporelles : LTL

Une chaîne de Markov permet donc de représenter et simuler l'évolution d'un système, lorsqu'il est soumis à des évènements extérieurs probabilistes. L'étape suivante est de vérifier automatiquement des propriétés sur ce système : dans l'exemple du jeu d'échec, il s'agirait de déterminer toutes les séquences pouvant mener à un échec et mat ; dans le cas de la figure 3 il serait intéressant de trouver les chemins menant à l'état "STOP", c'est-à-dire les chemins tels que la personne tombe à l'eau. En exprimant des propriétés à vérifier sur les chemins d'exécution possibles, sous forme d'une logique modale, un solveur permettra de dire immédiatement si celle-ci est valide ou non, voire fournira un contre-exemple violant cette propriété.

Une logique modale [Che80] est un langage mathématique basé sur la logique du premier ordre, à laquelle elle ajoute de nouveaux opérateurs et leurs règles de combinaison. Une formule de prédicat du premier ordre a en effet des opérateurs tels que "Et", "Ou", "Non", "Implique" ainsi que des quantificateurs tels que "Quel que soit" et "Il existe" ; en combinant ces opérateurs, on obtient une formule qui vaut "Vrai" ou Faux" lorsqu'elle est évaluée sur un ensemble d'objets.

Une logique modale ajoute de nouveaux opérateurs tels que "Il est possible" ou "Il est nécessaire". Il ne s'agit plus seulement d'évaluer la formule sur un ensemble d'objets, mais de l'évaluer sur un ensemble de mondes possibles, tels que les futurs possibles pour les Logiques Temporelles, les croyances possibles pour les logiques épistémiques ou doxastiques, ou encore les conséquences possibles de nos actions pour les logiques déontiques.

Les logiques modales utilisées dans la validation formelle de systèmes critiques sont des **Logiques Temporelles**, telles que la logique LTL (Linear Temporal Logic) [Pnu77] : les Logiques Temporelles sont ainsi construites autour de formules booléennes, étant vraies ou fausses dans chaque état, et d'opérateurs modaux. Par exemple, LTL dispose des opérateurs :

- X pour "next", où Xf est vraie si au prochain pas de temps f est vraie.
- U pour "until", où fUg est vraie si f est vraie au moins jusqu'à ce que g soit vraie (et il existe un pas de temps où g devient vraie).

f et g sont ici des formules booléennes, qui peuvent potentiellement être d'autres formules écrites en LTL.

Model-Checking probabiliste : PCTL

Il existe, comme nous le verrons dans l'analyse étendue de l'état de l'art, de nombreuses Logiques Temporelles permettant de réaliser des analyses très différentes. Nous pouvons en présenter une en particulier qui a pour spécificité de s'intéresser aux systèmes critiques subissant des évènements redoutés probabilistes : le langage PCTL (Probabilistic real-time Computation Tree Logic) [HJ94] permet d'exprimer des propriétés complexes sur les chemins d'une chaîne de Markov. Il repose sur une syntaxe proche de LTL, définissant des formules à partir des opérateurs de logique du premier ordre, de fonctions "étiquettes" booléennes associées à chaque état et de l'opérateur temporel Until.

Cet opérateur est similaire à l'opérateur Until de LTL, mais en ajoutant des probabilités : il exprime le fait que la probabilité d'arriver dans un certain ensemble d'états, avant un certain temps et en gardant une certaine propriété vraie tout le long du chemin, soit supérieure ou inférieure à une valeur donnée. Ainsi $fU^{H}_{\leq 0.5}g$ signifie qu'on cherche les chemins tels que f soit vraie jusqu'à ce que g soit vraie et où g devient vraie avant un horizon de H étapes ; on souhaite alors vérifier que la probabilité d'avoir un tel chemin soit inférieure à 0.5.

À titre d'exemple, sur la figure 3 il est possible de vérifier si la personne avançant aléatoirement sur le pont a une chance raisonnable d'atteindre l'autre côté sans tomber à l'eau, c'est-à-dire de vérifier par exemple si la probabilité d'atteindre l'état STOP avant 5 coups est inférieure à 0.01 : $(true)U^{5}_{\leq 0.01}(STOP)$. De manière similaire, il est possible de vérifier si la probabilité d'atteindre l'état STOP sans passer par la voie de gauche est supérieure à 0.05 : $(\neg G)U^{\infty}_{\geq 0.05}(STOP)$.

PCTL permet en particulier d'exprimer les contraintes suivantes :

- **Obligation** : $(true)U^{\infty}_{=1}g$ Le système doit toujours atteindre un état où g est vraie.
- **Interdiction** : $(true)U^{\infty}_{=0}g$ Le système ne doit jamais atteindre un état où g est vraie.

– **Antériorité** : $(\neg f)U_{=0}^{\infty}g$ Le système n'a pas le droit d'atteindre un état où g est vraie avant d'atteindre un état où f est vraie.

Définition de l'opérateur Until

Le langage PCTL permet aussi d'exprimer naturellement tous les autres opérateurs classiques de Logique Temporelle (LTL, CTL, ...) [HJ94], tels que Always (A), Exists (E), Globally (G) ou Finally(F), en se basant sur l'opérateur Until (dont cette version particulière est appelée "Strong Until" dans la littérature). Cette section rappelle donc la définition de cet opérateur avant d'exprimer la syntaxe générale du langage.

Définition 15 (*Sémantique de l'opérateur (Strong) Until*)

Soit S un espace d'états dénombrable,
Soit M une chaîne de Markov sur l'espace S,
Soit $s \in S$ un état particulier,
Soit Φ_s l'ensemble des chemins obtenus en débutant en s et en parcourant la chaîne de Markov M, avec pour $\phi \in \Phi$ un chemin donné :
- *$\phi(i)$ est l'état numéro i du chemin ϕ.*
- *ϕ_i est le sous-chemin de ϕ débutant en $\phi(i)$.*

Soit $t \in \mathbb{N} \bigcup \{\infty\}$ un horizon temporel,
Soit $\diamond$ un opérateur parmi $\{<,\leq,\geq,>\}$,
$U_{\diamond p}^{\leq t} : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}^S$ est l'opérateur de Logique Temporelle strong until*, défini par :*

$$\forall f : S \to \{0,1\}, \forall g : S \to \{0,1\}, fU_{\diamond p}^{\leq t}g = 1 \Leftrightarrow$$

$$\Pr\Big(\exists \phi \in \Phi_s, \exists 0 \leq i \leq t : g(\phi(i) = 1, \forall 0 \leq j < i, f(\phi(j)) = 1\Big) \diamond p$$

Cette définition formelle n'est cependant pas utilisable pour calculer explicitement si une formule PCTL est vraie ou fausse (i.e. sa valeur de vérité). Il est cependant possible de s'appuyer sur le résultat suivant pour calculer cette valeur de manière récursive :

Lemme 2 (*Calcul de l'opérateur Until*)

Soit S un espace d'états dénombrable,
Soit A un espace d'actions dénombrable,
Soit $T : S \times A \times S$ la fonction de transition associée à un processus de décision markovien,
Soit $\pi : S \times A \to [0;1]$ une politique stochastique,
Soit $s \in S$ un état particulier,
Soit Φ_s^π l'ensemble des chemins obtenus en débutant en s et exécutant π, avec pour $\phi \in \Phi$ un chemin donné :

- *$\phi(i)$ est l'état numéro i du chemin ϕ.*
- *ϕ_i est le sous-chemin de ϕ débutant en $\phi(i)$.*

Soit $t \in \mathbb{R}^+ \bigcup \{\infty\}$ un horizon temporel,
Soit $f : S \to \{0,1\}$ et $g : S \to \{0,1\}$ deux formules booléennes sur les états
On pose :

$$[P_f^g]_t^\pi(s) = \Pr\left(\exists \phi \in \Phi_s^\pi, \exists 0 \leq i \leq t : g(\phi(i) = 1, \forall 0 \leq j < i, f(\phi(j)) = 1\right)$$

$$T^\pi(s,s') = \sum_{a \in A} \pi(s,a) T(s,a,s')$$

$[P_f^g]_t^\pi(s)$ est alors solution du système d'équations dynamique :

$$[P_f^g]_t^\pi(s) = \begin{cases} 1 & \text{si } g(s) = 1 \\ 0 & \text{si } (g(s) = 0) \wedge (f(s) = 0 \vee t = 0) \\ \sum_{s' \in S} T^\pi(s,s') [P_f^g]_{t-1}^\pi(s') & \text{sinon} \end{cases}$$

Détails : Notons que cette définition repose sur des termes qui ne seront définis que dans des paragraphes ultérieurs, lors des rappels sur les processus décisionnels markoviens ; il aurait été possible de définir ce calcul de l'opérateur Strong Until uniquement en nous basant sur la définition d'une chaîne de Markov, mais le calcul de cet opérateur sur une politique solution d'un processus décisionnel markovien est suffisamment spécifique pour justifier le fait de mettre en avant cette version.

La preuve de la convergence de ce système d'équations dynamiques est relativement simple, puisque pour un état s donné la probabilité est croissante et bornée par 1 ; des détails sont fournis dans la littérature abondante [HJ94]. La valeur de vérité de l'opérateur Strong Until peut donc s'obtenir en comparant $[P_f^g]_t^\pi(s)$ avec la probabilité p de l'opérateur.

Ceci permet ainsi d'obtenir la définition syntaxique suivante, à partir de laquelle il est possible d'exprimer tous les autres opérateurs :

Définition 16 (*Syntaxe PCTL*)

La syntaxe d'une formule PCTL est définie par induction comme suit :

- *Toute proposition atomique (label booléen) est une formule PCTL.*
- *Si f_1 et f_2 sont des formules PCTL, alors $\neg f_1$ et $(f_1 \wedge f_2)$ sont des formules PCTL.*
- *Si f_1 et f_2 sont des formules PCTL, t est un entier positif ou ∞, p est un nombre réel avec $0 \leq p \leq 1$, $\diamond$ un opérateur parmi $\{<,\leq,\geq,>\}$, alors $f_1 U_{\diamond p}^{\leq t} f_2$ est une formule PCTL, où*

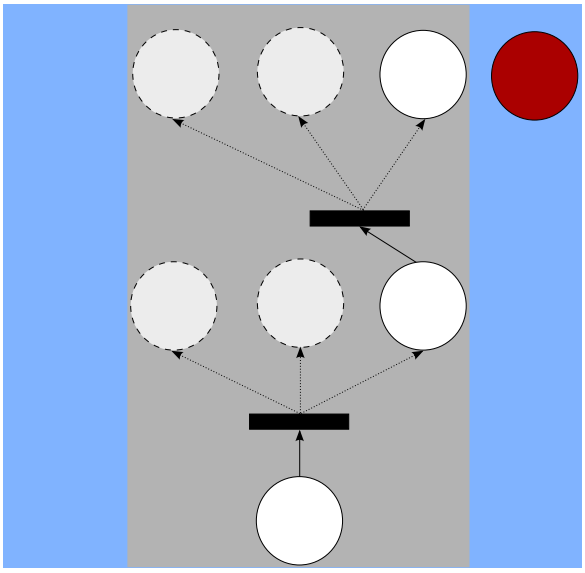
$$U_{\diamond p}^{\leq t} : \{0,1\}^S \times \{0,1\}^S \to \{0,1\}^S$$

est l'opérateur de Logique Temporelle strong until.

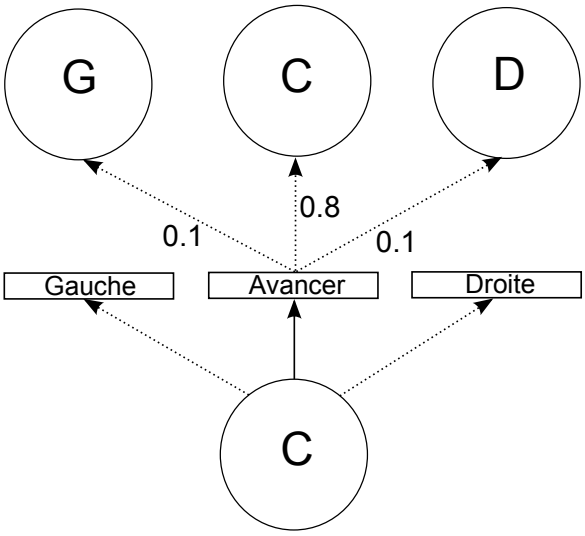
I.3.3 Processus décisionnels markoviens

Exemple académique

L'autre composante des problèmes de conception sûre et optimale, la *décision*, repose aussi sur des chaînes de Markov : considérons à nouveau l'exemple de l'homme traversant le pont, en modifiant le modèle de façon à ce qu'à chaque étape de temps, il peut choisir librement d'aller tout droit, aller à sa gauche ou à sa droite (figure 4). Le résultat de cette action en revanche est toujours stochastique : il a une certaine probabilité d'aller effectivement dans la direction choisie (0.8) et une probabilité d'aller légèrement à droite (0.1) ou à gauche (0.1) de la direction choisie.



(a) chaîne de Markov contrôlable : l'utilisateur cherche la meilleure action pour ne pas tomber à l'eau



(b) On choisit une action dans chaque état, dont le résultat est stochastique

FIGURE 4 – Exemple d'un Processus Décisionnel Markovien

Ce nouveau modèle correspond alors à une **chaîne de Markov contrôlable**, c'est-à-dire une chaîne de Markov pour laquelle il est possible de choisir à chaque temps entre plusieurs alternatives, par exemple de choisir la meilleure action adaptée à une situation pour ne pas se retrouver à l'eau : si l'homme est à droite, il choisira de revenir vers le centre. En revanche, l'effet de l'action est aléatoire : si l'homme choisit d'avancer tout droit, il est possible qu'il se passe quelque chose qui fasse qu'il arrive finalement à droite ou à gauche.

Le principe de *meilleure action* est exprimé par une récompense, positive ou négative, obtenue lorsque l'action a été effectuée : un agent dans un état s et effectuant l'action a recevra une récompense $R(s, a)$. Le but de l'agent (ici l'homme sur le pont) est d'obtenir la meilleure somme totale de récompense, ce qui peut par exemple nécessiter de prendre une décision coûteuse à l'instant présent (comme réparer un système) pour optimiser le résultat final.

Définition formelle

Le cadre des **Processus Décisionnels Markoviens** (MDP [Put94]) définit formellement ce type de modèles.

Comme représenté sur la figure 5, dans chaque état l'agent choisit une action parmi un ensemble d'actions possibles ; la liste de ces actions est connue et l'état est considéré entièrement connu (on dit qu'il est entièrement observable). Chaque action a un résultat stochastique : après avoir exécuté l'action, le système peut se retrouver aléatoirement dans certains autres états. Ces probabilités de passage d'un état à un autre sont représentées par une fonction de transition $T : S \times A \times S \to [0; 1]$, telle que $T(s, a, s')$ soit la probabilité que le système se retrouve dans l'état s' après que l'agent ait appliqué l'action a dans l'état s.

Passer ainsi d'un état à un autre donne une Récompense à l'agent, qui peut alors observer le nouvel état du système et choisir la prochaine action à effectuer.

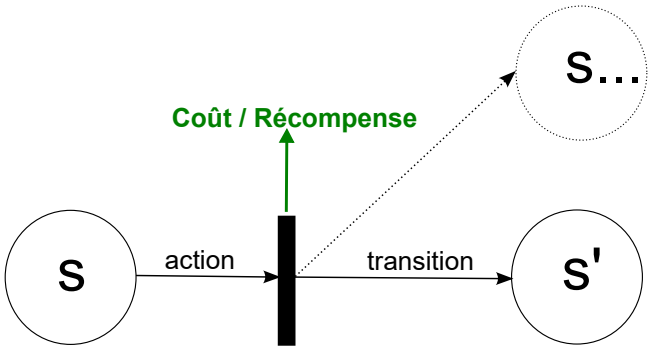


FIGURE 5 – Processus Décisionnel Markovien : états, actions causant des transitions non-déterministes, et récompense

Définition 17 (*Processus de décision markovien*)

Un processus de décision markovien est un vecteur (S,A,R,T,I) où :

S *est un espace d'états dénombrable*

A *est un ensemble dénombrable d'actions*

R : $S \times A \to \mathbb{R}$ *est une fonction de récompense*

T : $S \times A \times S \to [0;1]$ *est une fonction de transition, donnant une probabilité de transition*

I : $S \to [0;1]$ *est une distribution de probabilité initiale*

Chercher une solution à un MDP revient donc à chercher une politique : il s'agit d'associer à chaque état une action à effectuer. On parle de politique stationnaire si cette association ne change pas avec le temps. On distingue usuellement plusieurs types de politiques (Définition 18 page 24) :

Définition 18 (*Politiques*)

Soit un processus de décision markovien (S,A,R,T,I). On définit :

- Π^{SD} *L'espace des politiques stationnaires déterministes* $\pi : S \to A$.
- Π^{SR} *L'espace des politiques stationnaires stochastiques* $\pi : S \times A \to [0,1]$.

On dit ainsi d'une politique qu'elle est ***déterministe*** *si elle associe une seule action à chaque état. On dit qu'elle est* ***stochastique*** *ou aléatoire (randomized en anglais) si elle associe plusieurs actions à chaque état selon une distribution de probabilité. On dit qu'elle est* ***non-déterministe*** *si elle associe plusieurs actions à chaque état sans considération de probabilité.*

On a : $\Pi^{SD} \subset \Pi^{SR}$

On dit d'une politique qu'elle est ***markovienne*** *ou histoire-indépendante lorsque la distribution de probabilité ou l'action choisie ne dépend pas de l'histoire du système. On dit qu'elle est* ***non-markovienne*** *ou histoire-dépendante lorsque l'histoire du système influence la décision dans l'état courant.*

Détails : Les politiques stochastiques se différencient donc des politiques déterministes en ce qu'elles sont appliquées après un tirage aléatoire : dans un état donné s, l'utilisateur a une probabilité $\pi(s,a)$ de choisir d'appliquer l'action a.

Comme le montre la figure (Figure 6 page 25), les politiques non-markoviennes aléatoires sont les plus générales, tandis que les politiques markoviennes déterministes sont les plus spécifiques.

Précisons que ces définitions se limitent aux politiques stationnaires, c'est-à-dire aux politiques ne prenant pas en compte le temps où elles sont appliquées. Ceci revient à supposer que la politique est décidée une fois pour toute au début de la simulation, que la dynamique du monde (la fonction de transition) est elle aussi stationnaire et que les récompenses (donc la meilleure politique) ne sont pas susceptibles de changer en fonction du temps.

Notons que la distinction entre politiques markoviennes et politiques non-markoviennes est en pratique difficile à évaluer : il est en effet toujours possible de modifier l'espace d'états pour rendre les politiques markoviennes, en ajoutant à l'espace d'états des variables mémorisant l'histoire du système. À une politique donnée, il est donc difficile d'évaluer dans quelle mesure elle est indépendante ou non du temps : par exemple, si une des variables utilisées passe "vraie" après une action particulière, cette politique aura une forme de dépendance dans l'historique du système ; cependant, un tel ajout était peut-être nécessaire pour décrire la dynamique... Cette notion de markovien/non-markovien n'est donc intéressante que pour évaluer dans quelle mesure les décisions prises par une politique doivent changer en fonction de certaines variables clés.

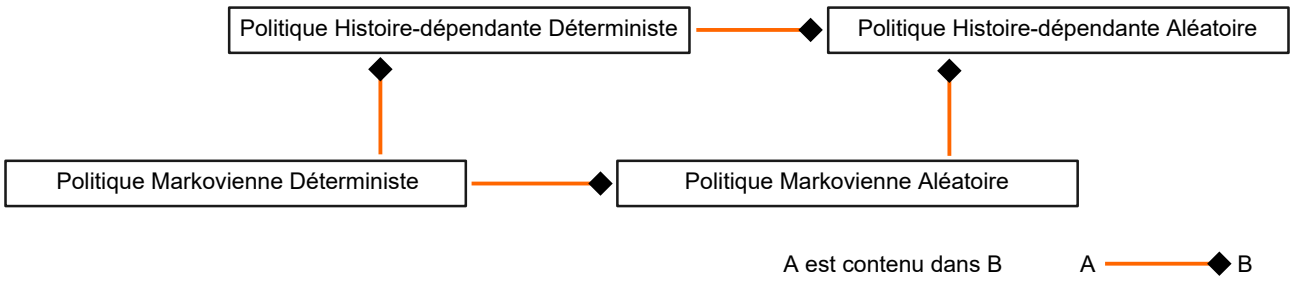


FIGURE 6 – Ensemble des politiques markoviennes/histoire-dépendantes déterministes/stochastiques.

Enfin, la solution d'un MDP est définie à partir de la fonction de valeur, qui représente le gain total espéré en appliquant une politique : dans un état s, $V^\pi(s)$ représente la somme des différentes récompenses que le contrôleur du système s'attend à avoir en continuant d'appliquer la politique π. Pour s'assurer que cette somme ne soit pas infinie, la fonction de valeur ajoute une dévaluation : les récompenses les plus lointaines comptent moins que les récompenses les plus immédiates.

Définition 19 (*Fonction de valeur dévaluée*)

Soit un processus de décision markovien (S,A,R,T,I),
Soit $\pi \in \Pi^{SR}$ une politique stochastique,
Soit $\gamma \in [0;1]$ un facteur de dévaluation,

La fonction de valeur, ou récompense totale espérée avec dévaluation, de la politique π est définie par :

$$\begin{aligned}\forall s \in S, V_\gamma^\pi(s) &= E_s^\pi \left\{ \sum_{t=0}^{\infty} \gamma^t R_t \middle| s_0 = s \right\} \\ &= \sum_{t=0}^{\infty} \gamma^t \left(\sum_{s_t \in S} \Pr(X_t = s_t | s_0 = s) \left(\sum_{a_t \in A} \pi(s_t, a_t) R(s_t, a_t) \right) \right) \\ &= \sum_{a \in A} \pi(s,a) \left(R(s,a) + \gamma \sum_{s' \in S} \left(T(s,a,s') V_\gamma^\pi(s') \right) \right)\end{aligned}$$

où E désigne l'espérance mathématique, (R_t) est la variable aléatoire représentant la récompense acquise exactement au temps t et (X_t) est la chaîne de Markov des états parcourus.

Détails : Le paramètre γ de dévaluation est à la fois un outil purement mathématique et un paramètre réel : lorsqu'il est strictement inférieur à 1, il permet de s'assurer que la fonction de valeur dévaluée (aussi appelée

"actualisée" dans la littérature) existe toujours même en horizon infini. Il a aussi une interprétation matérielle, puisqu'il correspond à la valeur qu'on accorde aux récompenses qui auront lieu. Ainsi, plus γ est proche de 0 et moins on accorde d'importance aux récompenses que le contrôleur du système s'attend à recevoir dans un futur lointain : une valeur de $\gamma = 0$ est le cas extrême où seule la récompense immédiate, reçue à l'état suivant, est prise en compte. D'autres formes de fonctions de valeurs ont été utilisées dans la littérature, par exemple des fonctions de valeurs sans dévaluation ou effectuant une moyenne sur un nombre N d'états successifs [Put94].

La solution d'un MDP est donc la politique ayant la meilleure valeur à l'état initial. Dans le cadre des MDP classiques, une telle solution existe toujours sous forme d'une politique déterministe.

Les MDP sont la base de nombreux autres modèles, permettant notamment de gérer différents types de contraintes, différents types d'actions (temps de décision continu, actions concurrentes,...) ou différents espaces d'états (espace continu, espace partiellement observable,...).

CHAPITRE II

ÉTAT DE L'ART

Sommaire

Bien que nous n'ayons pas encore défini notre problème en termes mathématiques et non ambigus, il est pertinent de démarrer cette étude par un état de l'art des technologies et méthodes pouvant contribuer à notre problème de conception sûre et optimale.

Cet état de l'art étend l'introduction du positionnement de cette étude vis-à-vis des différents domaines existants dans la littérature, que nous avons débuté au chapitre précédent. L'objectif de cette partie est ainsi de débattre de l'adéquation des études antérieures à notre problème, en abordant les points pouvant contribuer à notre étude ou pouvant nécessiter des adaptations particulières. Chaque thématique bibliographique est suivie d'un résumé des questions qui étaient laissées ouvertes en amont de notre étude.

L'objectif de ce chapitre n'est pas de choisir une méthode de résolution pour notre problème : même si à partir d'un état de l'art il est possible d'avoir une intuition du cadre mathématique qui serait le plus approprié, il nous faudra néanmoins dans une partie ultérieure analyser et définir le problème de conception sûre et optimale, avant de déterminer quel cadre de la littérature pourrait être le plus approprié. Lors des parties suivantes, nous proposerons à plusieurs étapes un état de l'art spécifique pour étudier la pertinence des méthodes existantes pour un problème particulier, ainsi que pour détailler les inspirations et pré-requis que nous utiliserons dans nos méthodes de résolutions.

Le choix entre plusieurs méthodes existantes sera donc laissé au sein de chaque partie au fur et à mesure de la rencontre de problèmes à résoudre. La lecture de ce chapitre n'est ainsi pas nécessaire à la compréhension des chapitres suivants.

L'apport de ce chapitre est donc de justifier que la question que nous nous posons, celle du développement d'un processus outillé pour la conception sûre et optimale, est la bonne question : il s'agit de la bonne question si au regard des travaux précédents nous disposons de suffisamment d'outils pour aborder notre problème, et si évidemment notre problème ne se ramène pas de manière triviale à des travaux déjà existants.

II.1 État de l'art sur la conception de systèmes auto-adaptatifs

Comme nous l'avons évoqué en introduction, l'une des premières communautés scientifiques qui est approchée par notre étude est celle de **l'aide à la décision**, et plus précisément de la **planification de décisions** [1].

L'objectif de l'aide à la décision est d'assister un utilisateur dans le choix d'une décision ; ceci peut être effectué de plusieurs manières très différentes, et le choix de la méthode dépend du type de modèle de décision et d'environnement. Dans la planification, on cherche à assister l'utilisateur pour prendre des décisions successives ; cet ensemble de décisions est appelé un **plan**. Il est alors possible de conseiller l'utilisateur sur les bons et les mauvais plans, la notion de bon et mauvais étant exprimée au moyen d'une contrainte ou d'une préférence. À nouveau, selon le modèle de contrainte ou de préférence choisi, on obtient des classes différentes de problèmes dans la littérature.

II.1.1 Cadres de planification

La planification est un domaine qui a déjà eu plusieurs succès au niveau industriel par le passé ; il s'agit d'un des domaines les plus largement associés aux technologies d'intelligence artificielle, et utilisés dans des applications courantes telles que les navigateurs GPS pour voitures, la robotique autonome par exemple pour les rovers envoyés sur mars, ou encore l'intelligence artificielle dans les jeux d'échec.

Sur le plan mathématique, la planification est apparue initialement comme extension des techniques d'exploration et de recherche, en particulier de recherches dans des graphes tels que l'algorithme de Dijkstra [Dij59]. Ces techniques de recherche basiques permettent de trouver le meilleur chemin (par exemple le plus court) permettant d'atteindre un but, en explorant un espace de chemins de manière exhaustive. Chacune des étapes de ce chemin correspond bien à une décision qu'il faut prendre pour atteindre un but, et le but est atteint après qu'on ait effectué un plan, c'est-à-dire une succession de décisions.

Très vite, ces techniques ont été remplacées par des algorithmes n'explorant par exhaustivement l'ensemble des états, le plus connu étant l'algorithme A* [HNR68] qui utilise une connaissance supplémentaire, appelée heuristique, dont on peut disposer sur le système. Plus précisément, ce type d'algorithme est *guidé* dans sa recherche par une information supplémentaire, contrairement aux algorithmes qui cherchent à atteindre un objectif en aveugle en essayant plusieurs voies possibles sans informations. De telles techniques permettent de réduire largement le temps de calcul, tout en garantissant l'optimalité du résultat, sous condition qu'il soit possible d'obtenir cette heuristique.

D'autres algorithmes utilisent en même temps la structure même de l'espace d'états pour optimiser la recherche ou le calcul ; un exemple est l'algorithme de Tarjan [Tar72] pour la recherche des Composantes Fortement Connexes [2]. Cette idée d'exploiter la structure des données pour aider la recherche du plus court chemin, éventuellement en convertissant cette structure en heuristique exploitable, a donné naissance à la **planification classique**, dont fait partie A*.

De nombreux algorithmes existent sur la planification classique [NGT04], on pourra mentionner en particulier les méthodes de **programmation dynamique**, de **recherche gloutonne**, de **recherche arborescente**, de **recherche heuristique** ou encore de **recherche locale**[GNT04]. Beaucoup se basent sur une représentation particulière des données appelées STRIPS : il s'agit d'un langage formel décrivant les objets sous forme de prédicats, c'est-à-dire de propriétés qu'il est possible d'ajouter ou d'enlever à un objet. En transformant une décision (appelée action) en un ensemble d'éléments à ajouter et à enlever à l'état actuel, il est possible d'obtenir des algorithmes très efficaces en termes de temps de calcul tels que GRAPHPLAN [BF97]. Ce gain d'efficacité semble naturel, puisque la représentation en STRIPS apporte par elle-même des informations supplémentaires : le but est naturellement divisé en sous-buts, c'est-à-dire en éléments précis qu'il nous faut obtenir à partir de l'état actuel, qui eux-mêmes peuvent être divisés en buts intermédiaires qu'il nous faut remplir successivement.

Cette représentation en prédicat porte la plupart des modèles et techniques de planification actuels. Cependant, d'autres formalismes existent tels que les travaux sur "Planning as Model-Checking" de

1. Decision-theoretic planning
2. Strongly Connected Components en anglais

Giunchiglia et Traverso [GT00], qui rapprochent planification et Model-Checking CTL. L'objectif est d'utiliser les algorithmes existant dans la validation formelle pour résoudre des problèmes de planification, ce qui nécessite d'exprimer les problèmes de planification sous la forme d'un problème de Model-Checking. Giunchiglia et al. ont ainsi proposé un planificateur (MBP : Model Based Planner) basé sur l'outil NuSMV permettant de résoudre des problèmes de Model-Checking à l'aide d'une représentation symbolique de l'espace des états atteignables ; la représentation du problème s'appuie en particulier sur des OBDD (Ordered Binary Decision Diagram). Ces travaux ont pour avantage principal de permettre une plus grande expressivité en termes de construction de modèles que la planification STRIPS. Ils ont pour avantage secondaire d'utiliser des technologies existantes, et de rapprocher par là même deux communautés qui sont souvent perçues comme distinctes. Cependant, il n'est pas évident d'établir qu'ils apportent un gain de performance sur un problème donné par rapport à d'autres méthodes de représentation : le problème du Model-Checking peut être vu comme un problème plus large que celui de la planification, ce qui laisse à penser que des algorithmes dédiés à la planification peuvent s'avérer plus performants que des algorithmes plus généraux. De plus, si de nombreux travaux existent sur la planification STRIPS en environnement probabiliste, ou sur la planification sous contraintes, ceci n'est pas encore le cas pour les travaux de "Planning as Model-Checking". Pour notre cas d'étude, cela signifie que ces travaux ne peuvent être utilisés que si des méthodes de résolution du problème de conception sûre et optimale existent en se basant uniquement sur des techniques de Model-Checking.

Une autre approche intéressante consiste à transformer le problème de planification en un problème de résolution de contraintes. On peut en particulier noter les travaux de Pralet et al., [PVLI10] sur la synthèse de contrôleur à partir de la résolution d'un problème de **satisfaction de contraintes (CSP)**. Ils ont en particulier mis en valeur la grande expressivité permise par cette approche, qui permet de traiter de multiples domaines de la planification tels que la planification non-déterministe ou la planification en environnement non-observable (POMDP [Mon82]). À nouveau, l'un des intérêts principaux est de pouvoir réutiliser des algorithmes très performants qui ont été développés par le passé pour résoudre les problèmes CSP. Cependant, ces méthodes peuvent s'avérer limitées : le temps de calcul augmente de façon exponentielle en fonction du nombre de variables, et la traduction du problème de planification en un problème CSP résulte en l'utilisation de nombreuses variables pour exprimer chacun des états possibles. Ces méthodes semblent donc bien souvent moins efficaces (en termes de temps de calcul) que d'autres méthodes de planification, qui permettent d'explorer un ensemble réduit de l'espace des états ; elles peuvent cependant présenter une plus-value sur des problèmes pour lesquels l'ensemble des états doit être considéré, par exemple lorsque des contraintes doivent être garanties. Pour notre problème de planification sûre et optimale, ces méthodes présentent l'intérêt immédiat de mettre au même niveau optimisation et validité, ce qui semblerait à même de résoudre notre problème, comme nous le verrons dans la suite de l'état de l'art ; cependant, la limitation en termes de temps de calcul est potentiellement un blocage important pour que ces méthodes puissent être envisagées dans un cadre industriel.

II.1.2 Processus Décisionnels Markoviens

Parmi les cadres de planification existants, nous sommes plus particulièrement intéressés par les études portant sur la planification dans un environnement soumis à des évènements probabilistes. En effet, comme nous avons pu l'évoquer dans la présentation du contexte - et comme nous le verrons dans un chapitre ultérieur - nous souhaitons prendre des décisions optimales tout en réagissant à des évènements tels que des défaillances de composants ; pour ces défaillances, nous disposons d'une estimation de la probabilité de défaillance à chaque instant, ce qui correspond bien à un modèle d'environnement subissant des évènements probabilistes.

Le cadre mathématique de base est celui des **Processus décisionnels Markoviens (MDP)** [Put94]. Comme nous l'avons évoqué en introduction, les travaux sur les MDP sont nombreux et diversifiés, puisqu'il s'agit d'un des modèles qui a été utilisé avec succès dans certaines applications industrielles. Ce modèle a pour avantage de permettre des algorithmes ayant une complexité raisonnable, c'est-à-dire dont le temps de calcul augmente de manière raisonnable avec la taille du problème : il est possible de prouver [PT87] que la résolution d'un MDP est de complexité polynomiale (sur un

espace d'états énumérés). Les méthodes les plus simples pour résoudre un problème MDP sont celles de **Programmation dynamique**[3], tels que les algorithmes de Value-iteration ou Policy-iteration.

Ce résultat n'est cependant valable que pour la version classique des MDP : dans leur version Partiellement Observable (POMDP [Mon82]), on prouve que la résolution est de complexité PSPACE-complete. Bien que les méthodes de base pour résoudre des MDP soient déjà relativement performantes, des travaux ont depuis obtenu des résultats bien meilleurs, par exemple en s'inspirant des méthodes de recherche de Monte-Carlo pour l'algorithme UCT [BPW+12].

Cependant, l'axe principal de recherche sur les MDP a été de choisir une sous-classe de problèmes possédant une structure particulière, qui permet d'obtenir des algorithmes heuristiques bien plus efficaces en termes de temps de calcul. La classe de problème des **SSP, pour Stochastic Shortest Path problem**[Ber95] est devenue la cible de référence pour les planificateurs probabilistes : il s'agit de résoudre un MDP pour lequel un ou plusieurs états ont été identifiés comme "but" à atteindre. Une telle hypothèse permet ainsi de concentrer la recherche en direction de ce but, ce qui permet de se dispenser de parcourir une grande partie de l'espace d'états.

L'un des algorithmes de résolution notable pour cette classe SSP est l'algorithme LAO* [HZ01], qui utilise les principes de l'algorithme A* sur la recherche heuristique, tout en exploitant autant que possible la structure de boucle présente dans le problème. Cet algorithme et le cadre SSP ont cependant deux défauts majeurs : les hypothèses de SSP sont très restrictives, imposant par exemple que toutes les récompenses soient strictement positives ; et dès lors qu'on essaie d'alléger ces hypothèses, l'algorithme LAO* est susceptible d'explorer des parties superflues de l'espace d'états, ce qui augmente de façon conséquente le temps de résolution. Plus précisément, les deux hypothèses restrictives des SSP sont :

– le SSP doit avoir au moins une solution valide, atteignant le but avec une probabilité 1.
– Toute solution non valide doit impliquer un coût ∞.

Pour pallier la seconde limitation, on peut par exemple citer les travaux de Kolobov et al. [KMWG11] qui ont présenté un nouvel ensemble d'hypothèses moins restrictives pour le problème SSP. Ce nouveau cadre, appelé GSSP pour Generalized SSP, permet en particulier de traiter tous les types de récompenses. Bien que la résolution d'un GSSP nécessite un pré-traitement particulier pour détecter et éliminer des boucles éventuelles, la complexité (en termes de temps de résolution) demeure la même que pour le problème SSP.

Une autre piste possible permettant de pallier les limitations de SSP consiste au traitement des impasses (dead-end) lors de la résolution : en effet, sous des hypothèses allégées, un algorithme peut se retrouver bloqué dans l'exploration d'une impasse, puisque rien ne garantit dans la forme des récompenses que le chemin optimal en termes de coût est un chemin atteignant le but. Les travaux de Kolobov et al. [KMW12] ainsi que de Teichteil et al. [TKVI11] se sont attaqués à ce problème. Ceci nécessite en particulier de définir de nouvelles heuristiques, prenant en compte les impasses. Ces travaux sont différents dans les hypothèses choisies pour étendre SSP : Teichteil et al. se basent sur une version dévaluée (discounted) du problème SSP, montrant que cette dévaluation est nécessaire pour traiter la totalité des formes de récompenses souhaitées. Kolobov et al. se concentrent sur plusieurs classes de problèmes étendant SSP, tels que MAXPROB pour lequel l'objectif est de maximiser la probabilité d'atteindre le but, ou SSPUDE pour lequel une impasse n'est pas évitable mais est associée à un coût fini ou infini.

Ces approches ont pour mérite d'étendre le domaine des problèmes qu'il est possible d'exprimer, tout en utilisant des techniques heuristiques efficaces pour les résoudre. Cependant, certains problèmes ne peuvent pas être exprimés sous la forme d'un (G)SSP ; c'est en particulier le cas des problèmes de synthèse de contrôleur valide pour des contraintes de Logiques Temporelles, comme nous le verrons par la suite.

Parmi les extensions notables des MDP, qui ne peuvent pas être ramenées trivialement à un SSP, on notera par exemple les travaux sur les MDP compétitifs [RCPF02], dans lesquels plusieurs acteurs prennent des décisions et pour lesquels la fonction de récompense de chaque acteur dépend aussi des actions de l'opposant.

Les travaux de Hauskrecht et al. [HMPK+98] sont aussi intéressants, en ce qu'ils exploitent une structure différente des MDP : en définissant des macro-actions, vues comme des politiques locales à

3. Dynamic programming

certaines régions de l'espace d'états, il est possible d'obtenir des algorithmes de résolution efficace en termes de temps de calcul.

Enfin, on pourra noter les travaux réalisés sur d'autres types d'incertitudes, par exemple les incertitudes sur la durée des actions. En particulier, Beaudry [Bea11] a développé un planificateur reposant sur une modélisation MDP avec actions concurrentes et des durées d'actions probabilistes, représentées par des variables continues.

Bien qu'ils ne semblent pas directement applicables à notre problème de conception sûre et optimale, ces deux travaux mettent en valeur la richesse des MDP, qui servent de base à de nombreux modèles. Ceci montre la maturité de ce cadre mathématique vis-à-vis des problèmes d'aide à la décision dans l'incertain.

II.1.3 Structure de l'espace d'états

Nous avons évoqué précédemment l'importance de la représentation des données dans la performance : en effet, dès lors qu'il est possible d'exploiter une structure particulière dans les données, un algorithme pourra disposer d'une heuristique le guidant dans sa recherche au sein de l'espace d'états.

Nous avons déjà parlé de l'importance des travaux réalisés autour du formalisme STRIPS, ainsi que des heuristiques qui continuent à être créées à partir de cette représentation [HBG05]. Dans le cadre des processus décisionnels markoviens, il est possible d'appliquer les mêmes principes, comme pour les travaux de Dean et Lin [DL95], utilisant la représentation en variables booléennes pour structurer l'espace d'états, et ramener la résolution du problème global en de multiples sous-problèmes.

Sur une formalisation plus proche de STRIPS, on notera les travaux de Younes et al. [YMS03] sur le langage **PPDDL (Probabilistic Planning Domain Definition Language)** : ce langage partant de PDDL, qui est un des langages de référence pour la planification classique exprimée sous forme de prédicats ; il y ajoute en particulier des actions stochastiques, ayant une certaine probabilité d'ajouter ou de retirer certains prédicats. PPDDL a été pendant plusieurs années le langage de référence pour les planificateurs probabilistes MDP/SSP, en particulier en raison de son utilisation dans la compétition internationale de planification IPC [ICA]. Bien qu'il ait été remplacé officiellement par le langage RDDL [San10] pour cette compétition, remplacement motivé par le fait que RDDL supporte nativement l'observabilité partielle, de nombreux planificateurs continuent toujours à le supporter pour la définition de problèmes.

II.1.4 Questions ouvertes en amont de notre étude

Au regard des travaux existants au début de cette étude (Figure 7 page 33), nous pouvons conclure que le cadre MDP est un cadre possible pour exprimer notre problème de conception sûre et optimale, voire un cadre adapté dès lors que le système présente un comportement probabiliste. Plus spécifiquement, nous pouvons en déduire que la plupart des techniques et planificateurs existants qui pourraient résoudre le type de problème que nous souhaitons traiter reposent sur le cadre MDP, qui aurait éventuellement été augmenté pour prendre en compte des contraintes sous une certaine forme. Dans la section suivante, nous allons ainsi regarder plus en détail les travaux réalisés sur des formats d'expression de contraintes.

Ce constat n'exclut pas, sans étude plus approfondie, que des travaux d'autres domaines puissent répondre à notre question. Nous débattrons de ce point dans un chapitre ultérieur, une fois que notre problème de conception sûre et optimale aura été défini sans ambiguïté. En revanche, le choix de s'orienter de façon préliminaire sur le cadre MDP correspond bien à l'un des objectifs de ce chapitre, qui est d'évaluer dans quelle mesure les technologies existantes sont assez matures pour porter une étude jusqu'à un niveau proche de l'industrie. Le cadre MDP, et les langages de représentation associés, nous apparaissent donc suffisamment développés pour justifier la création d'un processus outillé basé sur ces concepts.

La seule question demeurant ouverte vis-à-vis de ces technologies est ainsi la suivante : est-il possible d'adapter le cadre MDP pour prendre en compte des contraintes de sécurité ?

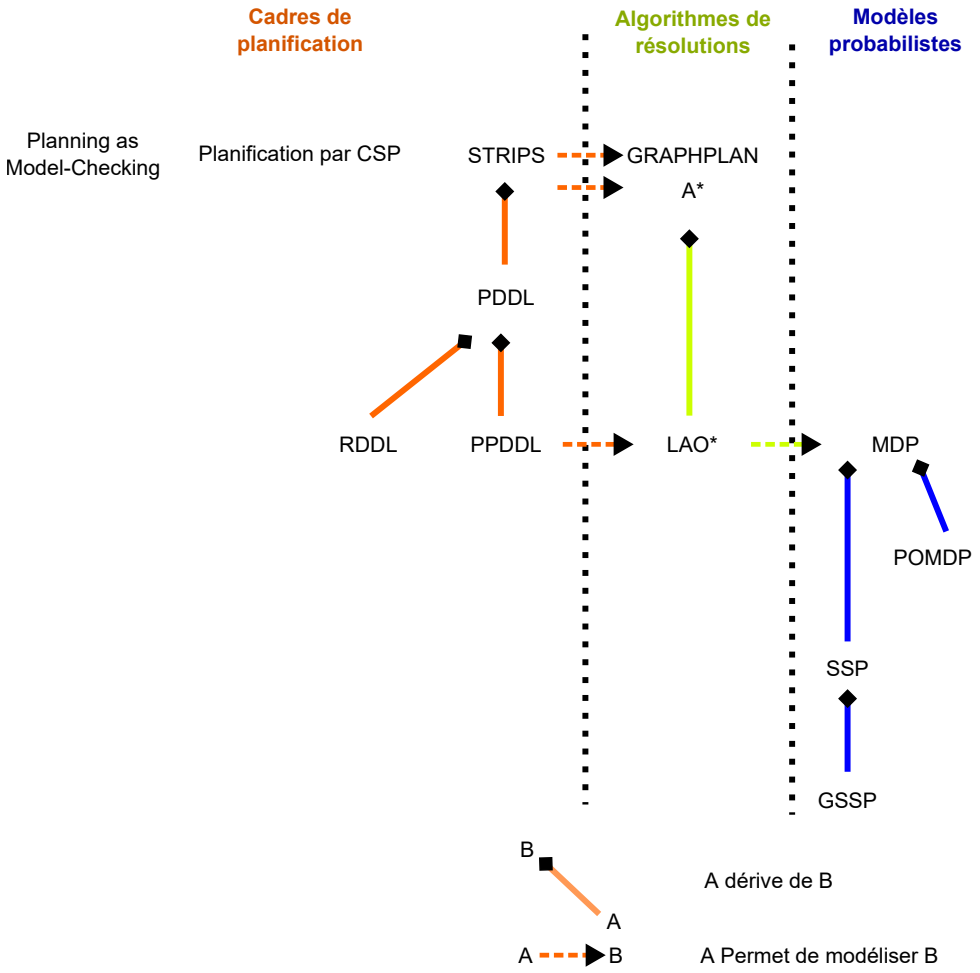


FIGURE 7 – État de l'art de la conception de systèmes autonomes, centré sur la planification en environnement probabiliste.

II.2 État de l'art sur l'analyse de systèmes critiques

La seconde communauté qui occupe une part toute aussi importante dans nos travaux est celle de l'analyse des systèmes critiques. Cette communauté regroupe les personnes et entités cherchant à définir la criticité d'un système, à l'évaluer, à proposer des outils permettant de l'assurer à un certain niveau et enfin à proposer un ensemble de règles de certification, qui sont gages de la qualité du système en termes de maîtrise des risques.

Dans l'aviation, nous avons déjà évoqué l'importance de la norme [ARP 4754] en matière de développement de systèmes avioniques. Pour garantir le respect de cette norme au regard de la complexité croissante des systèmes, de nouvelles techniques de génie logiciel sont régulièrement intégrées dans la certification des systèmes dès lors qu'ils sont jugés suffisamment matures. Parmi ces techniques de génie logiciel, les techniques de **méthodes formelles** en particulier sont considérées par beaucoup de spécialistes comme ayant atteint un niveau de maturité suffisant pour être intégrées aux processus de certification. Depuis 2012, elles sont par exemple mentionnées dans la norme DO-178C comme permettant de compléter (mais non remplacer) les phases de tests pour le développement des logiciels avioniques.

Ces techniques reposent sur une modélisation du système dans un langage formel, qui permet de vérifier automatiquement si l'architecture et la dynamique du système respectent un niveau suffisant de criticité. On parle de **Model-Based System Engineering** lorsque les modèles formels sont utilisés comme moyen de communication et d'information pour concevoir un système ; on parle de **Model-Based Safety Assessment** lorsque les modèles formels sont utilisés pour réaliser une analyse de risque de façon automatique à partir de modèles, par exemple en produisant des **arbres de fautes** ou une FMEA **(Failure Mode and Effects Analysis)**.

Pour exprimer cette notion de niveau suffisant de criticité, la majorité des études récentes de MBSA se basent sur les **Logiques Temporelles**.

II.2.1 Logiques Temporelles

Pnueli [Pnu77] en particulier a contribué à populariser l'utilisation des Logiques Temporelles pour la vérification de programmes. Dans ses travaux fondateurs, il définit une approche de vérification formelle basée sur la prise en compte de la dépendance temporelle entre évènements. Le temps est ici représenté comme une séquence d'états, et les dépendances entre états sont exprimées de manière mathématique sous la forme d'assertions logiques dans une logique modale, appelée depuis **Logique Temporelle Linéaire(LTL)**. Ces travaux étaient réalisés initialement dans le but de vérifier des conditions de séquentialité et parallélisme de programmes, mais ont été appliqués depuis à un vaste panel de domaines, théoriques et industriels.

La logique elle-même a dû être étendue pour prendre en compte d'autres modèles du temps. C'est en particulier le cas de la logique **CTL (Computational Tree Logic)**, proposée quelques années plus tard par Clarke et Emerson [CES86]. CTL est une Logique Temporelle où le temps est vu comme un arbre d'états successifs, ce qui représente le fait que le futur n'est pas déterminé. Une branche de l'arbre représente alors un futur possible. À partir de la logique CTL, on parle de **Model-Checking** pour désigner le fait de vérifier des propriétés sur un système ou un programme à partir de Logiques Temporelles. CTL permet d'exprimer des assertions telles que "lorsque toutes les variables sont positives, alors toutes exécutions possibles du programme ne mènent jamais à une division par 0". Un model-checker explorera alors de façon exhaustive toutes les branches possibles satisfaisant la condition initiale, et vérifiera que toutes les exécutions dans le futur respectent la seconde condition. Bien que CTL et LTL aient été développés de façon indépendante et parallèle, ils ont une grande proximité, en dépit de certaines propriétés qui ne peuvent être exprimées qu'en CTL ou en LTL.

Ce manque d'interopérabilité complète entre CTL et LTL a été à l'origine de la création de la logique **CTL***, permettant notamment de combiner librement des quantificateurs et des opérateurs temporels. Tout comme CTL, CTL* est une Logique Temporelle basée sur un arbre. On peut montrer trivialement que CTL* contient à la fois CTL et LTL.

En termes de complexité de vérification, Emerson [Eme90] a prouvé que la vérification d'une propriété LTL était de complexité PSPACE dans le cas général. Il a été prouvé par ailleurs que le

Model-Checking CTL* a la même complexité (PSPACE). Dans ses travaux, Emerson s'est intéressé à la comparaison de plusieurs Logiques Temporelles, y compris LTL et CTL, permettant de mettre en valeur ce qui est exprimable ou non dans une certaine logique. Il apparaît en particulier que de nombreuses Logiques Temporelles peuvent être construites sur des concepts différents, telles que des logiques centrées sur le passé au lieu du futur, des logiques prenant en compte des probabilités, ou encore des logiques permettant de raisonner sur l'état de la connaissance des systèmes réactifs.

Chacune de ces logiques gagne en expressivité par rapport aux logiques LTL et CTL ; nous allons à présent en détailler plusieurs qui nous semblent particulièrement pertinentes pour notre problème.

Baier, Katoen et Hermanns ont défini en 1999 [BKH99] la logique **CSL (Continuous Stochastic Logic)** permettant d'exprimer des propriétés dans un système à temps continu. Le modèle sous-jacent est celui des **Chaînes de Markov à Temps Continu**, dont nous rappellerons la définition par la suite. CSL est une logique particulièrement puissante ; elle est devenue la référence de nombreux outils pour le Model-Checking à temps continu, au même titre que LTL et CTL sont les références pour le Model-Checking à temps discret. Baier et al. ont proposé de nombreuses méthodes de résolution, montrant qu'il est possible de ramener les assertions CSL à des systèmes d'équations linéaires, pour lesquelles des méthodes très performantes existent. CSL a elle-même été étendue par la suite, en particulier avec l'ajout de récompenses par HaverKort et al. [HCH$^+$02] et avec l'ajout de probabilités par Baier et al. [BHHK03].

L'ajout de récompenses en particulier est intéressante pour notre problème, puisqu'elle permet d'imposer des vérifications sur des niveaux de performance dans le temps, en plus d'imposer des performances sur la séquentialité et l'atteignabilité des états. Ce type de considération nous rapproche des considérations que nous avons dans les systèmes critiques, pour lesquels nous souhaitons aussi évaluer les évolutions futures d'un système en fonction d'un niveau de performance, privilégiant par exemple les évolutions les moins onéreuses, ou les plus rapides, sous conditions évidemment qu'elles respectent aussi les autres contraintes de Logique Temporelle. De telles évaluations peuvent être exprimées en CSRL (Continuous Stochastic Reward Logic), qui est une logique se basant sur des **Modèles de récompense Markoviens**. Néanmoins, il est essentiel de noter que CSRL ne parle pas d'optimisation de récompense (cumulée), mais de vérification d'un niveau de performance : il ne s'agit pas de chercher la performance maximale, mais seulement de garantir qu'elle a en moyenne (probabiliste) un certain niveau. Cette distinction pourrait s'avérer bloquante pour notre problème, dans l'utilisation de logiques telles que CSRL, puisqu'il ne semble pas trivial d'adapter la vérification de propriétés CSRL à de la prise de décision sous contraintes CSRL.

L'ajout de probabilités à CSL est tout aussi intéressant : Baier et al. s'inspirent de plusieurs logiques existantes pour construire une logique évaluant des mesures de probabilités sur des ensembles de chemins dans une Chaîne de Markov à Temps Continu. Les auteurs proposent une méthode de résolution basée sur un système d'équation intégrale, ainsi qu'une réduction du problème au travers d'une bisimulation. L'intérêt pour notre problème est que les contraintes que nous devons prendre en compte sont de nature probabilistes : pour un système critique, il faut en général vérifier qu'une certaine séquence d'évènements, par exemple la perte d'une fonction de pilotage, n'arrive qu'avec une probabilité extrêmement faible ; c'est-à-dire qu'il faut vérifier que le système atteint un état défaillant avec une probabilité inférieure à un certain seuil. Au premier regard, ce type de contraintes que nous recherchons semble correspondre à celles qui sont exprimables avec cette extension de CSL.

Cependant, ces deux dernières extensions de CSL ont pour principale limitation d'augmenter de manière conséquente la complexité de la vérification, c'est-à-dire le temps de calcul moyen mis pour vérifier une assertion. Il est difficile d'évaluer *a priori* dans quelle mesure cette limitation rend inenvisageable l'utilisation de CSL pour notre problème, puisqu'il existe certains exemples dans la littérature où un algorithme à variables continues a de meilleures performances qu'un algorithme à variables discrètes. Cependant, nous pouvons identifier la cause principale de cette complexité : la logique CSL prend en compte un temps continu, ce qui nécessite un traitement intégral lors des cas les plus complexes. Il semble donc pertinent de s'intéresser en priorité aux modèles à temps discret, et de déterminer dans quelle mesure les modèles que nous souhaitons traiter nécessitent un temps continu.

La version probabiliste tire son inspiration de la logique **PCTL (Probabilistic real time Computation Tree Logic)** [HJ94], qui elle-même tire son inspiration de CTL et de travaux d'auteurs

ayant étendu CTL. Hansson et Jonsson définissent ainsi une logique exprimant des formules sur une **Chaîne de Markov à Temps Discret**, ainsi que des algorithmes permettant de vérifier ces formules en temps polynomial, dans la taille de la formule et dans la taille de la chaîne de Markov. Tout comme la version probabiliste ultérieure de CSL, les formules exprimées correspondent à la mesure de la probabilité d'occurrence de certaines formes de chemins. PCTL est devenue depuis une des logiques de référence pour les outils de Model-Checking probabiliste. Hansson et Jonsson ont montré que l'ensemble des opérateurs de CTL pouvaient être exprimés à partir des opérateurs de PCTL.

De nombreux travaux ultérieurs se sont basés sur PCTL pour proposer des extensions de la logique, des méthodes de vérification efficaces ou encore des problématiques de vérification différentes. Il est intéressant de noter par exemple les travaux de Brasfil et al. [BFKK08] sur la satisfiabilité de PCTL, c'est-à-dire le fait de trouver un modèle vérifiant une certaine formule PCTL. Ce problème est intéressant puisqu'il montre l'étendue des applications possibles pour PCTL, bien qu'il ne semble que difficilement praticable pour des cas industriels en raison de sa complexité : pour la satisfiabilité de CTL et PCTL, les travaux de Brasfil et al. montrent que la complexité est EXPTIME-complet.

En termes de performances, on pourra noter les méthodes récentes basées sur les **Strongly Connected Components (SCC)**[Tar72] tels que les travaux de Teichteil, Infantes et Seguin [TKIS11] : ces méthodes utilisent les algorithmes définis par Hansson et Jonsson en les combinant à une découverte à la volée des états d'un système, ainsi que des boucles présentes entre ces états. La découverte de boucle permet alors d'effectuer un ensemble de calculs locaux afin de rendre la vérification de la formule PCTL plus rapide. De telles méthodes, exploitant la structure des données, contribuent à rendre le model-ckecking probabiliste réalisable sur des modèles de taille industrielle avec les techniques de calcul de ces dernières années.

Enfin, parmi les extensions PCTL qui sont d'un intérêt particulier pour notre problème, on pourra noter les travaux récents de Yoo, Fitch et Sukkarieh [YFS12], qui augmentent l'expressivité du langage PCTL avec des contraintes sur des ressources, exprimées au moyen de variables dans $\mathbb{R}$. Ces contraintes expriment des seuils de performance ou de ressources qui doivent être respectés de manière ferme ou douce. Il s'agit encore une fois de concepts intéressants pour notre problème, puisque nous serons éventuellement amenés à devoir contraindre l'évolution d'un système en fonction d'un niveau de performance attendu. Cependant, ces méthodes se concentrent sur la validation de contraintes et non la synthèse de stratégie permettant de les valider, ou encore optimisant un niveau de performance. Il n'est donc pas évident sans une étude préliminaire de savoir s'il sera possible d'exploiter ces travaux en l'état, puisqu'il n'est pas trivial de passer de la vérification à la synthèse de contrôleur valide.

II.2.2 Outils de modélisation

Si l'avantage principal des techniques de Model-Checking est de proposer une analyse exhaustive des risques - ce qu'il est difficile de garantir par un processus manuel - leur inconvénient principal est le coût de construction du modèle. Les modèles que nous considérons sont des modèles logiques, c'est-à-dire qu'il n'est pas nécessaire de simuler précisément la valeur de chacun des signaux, mais simplement de se concentrer sur des notions telles que "transmis", "correct" ou "erroné" ; néanmoins il est nécessaire de créer ces modèles à partir de la connaissance des concepteurs du système, sous une forme qui puisse donc à la fois être facilement lisible par un être humain et lisible par une machine. On parle de **langage formel** pour décrire un langage lisible par une machine sans ambiguïté. Par exemple, une spécification écrite en texte libre dans un document peut comporter des ambiguïtés, alors qu'une formule mathématique n'en comporte pas. L'enjeu des outils assistant la modélisation formelle est donc de proposer des aides à un utilisateur pour saisir les données dans un langage formel ; ceci passe dans un premier temps par la définition de ce langage, qui agit comme pont entre l'utilisateur et la machine.

Arnold, Point, Griffault et Rauzy [APGR99] ont défini le langage Altarica dans cet objectif. Il s'agit d'un langage permettant de décrire le comportement d'un système complexe, constitué de plusieurs composants qui interagissent de façon dynamique. Le cœur de cette dynamique repose sur la notion de machine à état et de transition : chaque composant peut subir des évènements, le faisant changer d'état. Arnold et al. ont défini les règles syntaxiques et sémantiques de ce langage ; ils ont prouvé la cohérence du langage au travers de la définition d'une bisimulation, permettant de montrer que les

résultats de validation formels obtenus dépendent uniquement du comportement décrit dans le modèle, et non de la syntaxe choisie pour décrire chacun des composants.

Le langage Altarica a depuis donné naissance à de nombreuses dérivations, en particulier les versions Altarica Dataflow [Rau02] et Altarica 3.0 [Pro14] qui se sont concentrées sur les facilités de modélisation et de validation dans un contexte industriel. Altarica Dataflow en particulier a été intégré à des outils tels que Cecilia OCAS (Dassault Aviation), Simfia (EADS Apsys) ou Safety Designer (Dassault Systemes). Ces évolutions du langage apportent à la fois des avantages et des limitations, qui sont à prendre en considération en fonction du système que l'on souhaite modéliser [BBC+04]. Ainsi, Altarica Dataflow impose une restriction au langage pour en rendre la validation plus rapide en termes de temps de calcul ; cependant cette restriction dans la direction des flux de données rend impossible ou complexe la modélisation de certains systèmes, tels que des systèmes hydrauliques où il n'est pas aisé d'identifier une direction unique de propagation. À l'inverse, sans cette restriction, certains modèles sont trop complexes pour que les algorithmes de validation puissent les traiter en temps raisonnable. Il semble y avoir ainsi deux difficultés actuelles qui limitent l'acceptabilité de ce langage et des langages similaires par le milieu industriel : celle de trouver le juste milieu entre l'expressivité du langage et la facilité d'utilisation - ce qui comprend le temps de traitement des modèles - et celle de la reconnaissance par les processus de certification.

D'autres outils de Model-Checking existent, qui se sont attaqués à ces problèmes avec le même succès relatif. On citera par exemple le langage B [Abr96] et les suites d'outils Atelier B et RODIN, qui sont l'un des exemples d'application des méthodes formelles à des cas industriels concrets : le langage B a été utilisé pour certifier le logiciel embarqué de la ligne 14 du métro parisien. Ce langage a depuis été utilisé pour la certification d'autres lignes de métro. L'apport principal du langage B est qu'il permet de générer directement une partie du logiciel embarqué, dont il est prouvé qu'il vérifie des spécifications formelles. Cependant, son applicabilité à notre problème est limité puisqu'il ne permet pas de façon native de vérifier des contraintes probabilistes. On peut aussi citer des langages tels que ADL [MT00] ou SMV (ainsi que l'outil NuSMV permettant le Model-Checking [CCG+02]), et nous renvoyons un lecteur intéressé aux différentes études qui montrent les avantages et les inconvénients de ces formalismes.

Parmi les outils de Model-Checking probabiliste, on notera en particulier l'outil MRMC (Markov Reward Model Checker) [KZH+11], supportant le Model-Checking PCTL et CSL ainsi que les extensions de ces deux logiques avec des récompenses. Cet outil est intéressant dans la mesure où il offre des performances raisonnables pour la vérification de contraintes PCTL et CSL, ce qui correspond à une partie du problème que nous souhaitons résoudre. De façon similaire, l'outil et le langage PRISM [KP13] permet de réaliser de la vérification formelle probabiliste avec des techniques efficaces pour les logiques PCTL et CSL. Une partie de l'efficacité de ces techniques repose sur l'utilisation de structures de données adaptées telles que les Binary Decision Diagram [BRB91].

II.2.3 Questions ouvertes en amont de notre étude

Au regard des travaux existants au début de cette étude (Figure 8 page 38), nous pouvons en conclure que les technologies de vérifications formelles ont atteint une maturité suffisante pour être intégrées au sein de processus industriels. Nous pouvons aussi en conclure qu'aucun outil ne paraît avoir reçu d'approbation unanime de la part des concepteurs de systèmes ; ceci découle en particulier du fait que les deux questions suivantes ne semblent pas avoir reçu de réponse satisfaisante :

- Comment faciliter la saisie d'un modèle par des ingénieurs systèmes ?
- Quel est le niveau de modélisation nécessaire ? En particulier, à quel point peut-on réduire l'expressivité d'un langage tout en permettant à un utilisateur de saisir et vérifier les systèmes cibles ?

Il semble probable qu'il n'y ait pas de réponse unique à ces questions, qui dépendront des domaines d'applications industriels envisagés.

Si ces deux questions semblent principalement adressées aux outils et langages de modélisation, elles ont néanmoins un impact sur le modèle mathématique qui sous-tend la vérification. On peut de

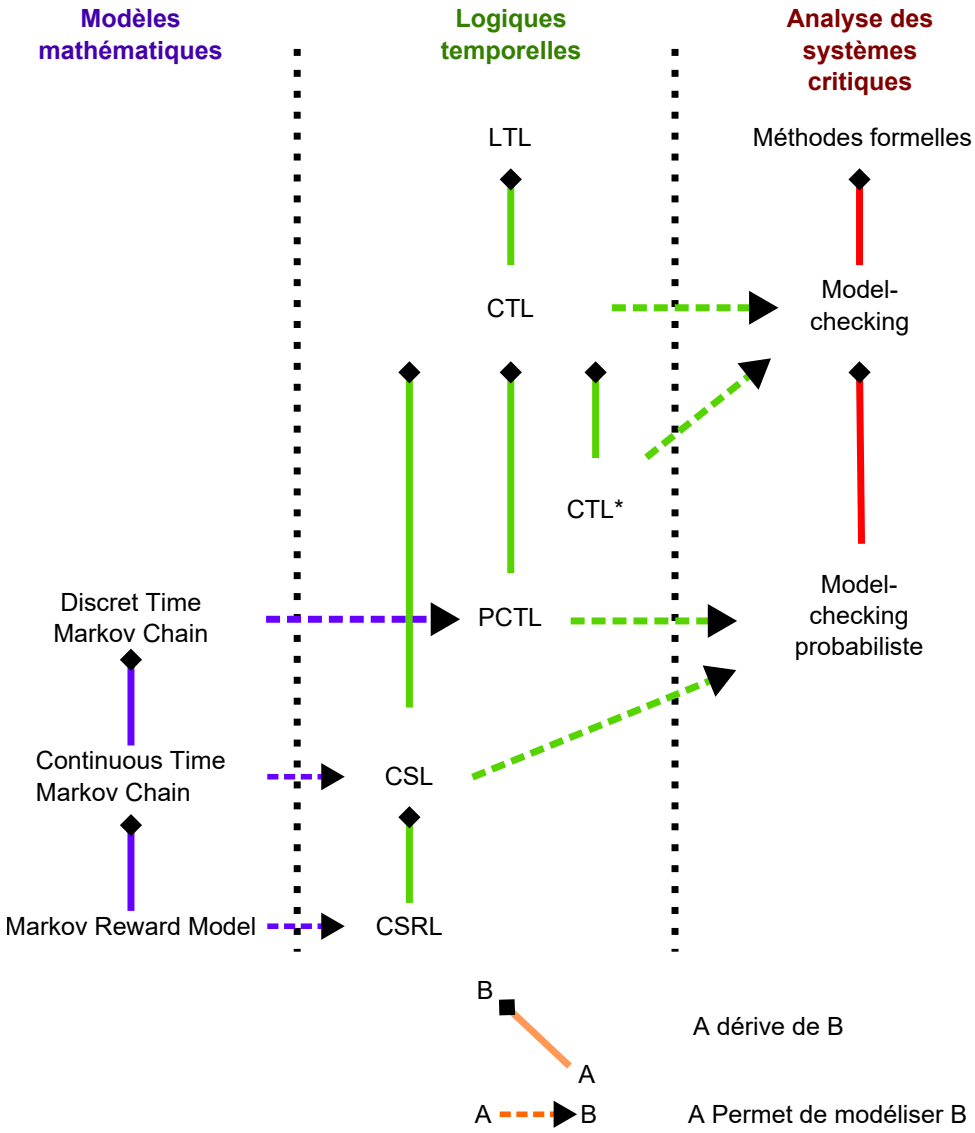


FIGURE 8 – État de l'art de l'analyse des systèmes critiques, centré sur les Logiques Temporelles pour l'expression de contraintes.

la même manière se poser la question de l'expressivité d'une Logique Temporelle choisie : dans quelle mesure est-il nécessaire de choisir des Logiques Temporelles complexes (CSL, PCTL, ...) pour exprimer les contraintes de notre problème, et dans quelle mesure peut on se limiter à des logiques plus simples et potentiellement plus faciles à vérifier ?

Il semble ainsi apparent que les technologies de modélisation formelles ont quitté le domaine de la recherche fondamentale, pour s'orienter vers une problématique d'industrialisation : la question n'est plus de fournir des réponses génériques à des problématiques théoriques, mais de faire face à l'acceptabilité par les industries ainsi qu'au problème de passage à l'échelle sur des données réelles.

II.3 État de l'art sur la conception de systèmes sûrs et optimaux

L'idée de contraindre la solution trouvée par un planificateur est naturelle dès lors qu'on cherche à appliquer l'aide à la décision à des cas industriels. En effet, contraindre une solution permet de garantir, dans une certaine mesure, que cette solution a un certain niveau de sécurité : en garantissant que le système ne sort jamais d'un cadre défini, on garantit que le système pourra être validé par les processus de vérification ou de certification. Lorsqu'une telle garantie est assurée lors de la planification, on parlera de synthèse de contrôleur sûre et optimale.

La forme des contraintes est l'élément déterminant : si les contraintes sont simples à exprimer et vérifier, il sera aisé de modifier le modèle MDP/SSP pour les prendre en compte ; lorsque les contraintes sont complexes, il faudra utiliser des techniques novatrices, éventuellement inspirées d'autres domaines tels que le Model-Checking.

Nous regarderons dans cet état de l'art plusieurs types de contraintes, et les modèles qui en découlent ; nous ne pourrons en revanche pas conclure sans étude supplémentaire sur le type de contraintes qui est applicable à notre problème.

II.3.1 Constrained Markov Decision Processes

Altman [Alt99] a été à l'origine d'une extension appelée **Constrained Markov Decision Processes (CMDP)**. Dans ce modèle, plusieurs récompenses (utilités) sont définies, correspondant à plusieurs objectifs : l'un de ces objectifs doit être optimisé, et les autres sont vus comme des contraintes, devant rester dans une certaine enveloppe ou atteindre un certain seuil. Altman propose plusieurs algorithmes de résolutions, en particulier par **programmation linéaire (LP)** [4] dans le cas où le nombre d'états est fini, et par passage dans un espace dual suivi d'une résolution LP dans le cas où ce nombre n'est pas fini - bien que des travaux se soient depuis intéressés à une approche par programmation dynamique [PM00]. L'un des concepts particulièrement intéressant qui est utilisé pour cette résolution est celui de **mesure d'occupation** : il s'agit d'une manière d'exprimer la probabilité qu'un état soit traversé, qui a une relation directe avec la décision optimale et valide qu'il faut prendre à chaque état. À travers cette mesure d'occupation, Altman montre des résultats intéressants sur la forme des politiques solutions, bien que nous reviendrons sur la définition des différents types de politiques existantes et sur ces résultats dans un chapitre ultérieur : il prouve ainsi que, pour les CMPD, les politiques optimales peuvent être aléatoires (randomized) et qu'il n'est pas possible de se limiter aux politiques déterministes lorsqu'on cherche l'optimalité.

On notera que ce résultat avait déjà été établi précédemment dans un cas plus restreint que CMDP : Beutler et Ross [BR85] ont réalisé des travaux se basant sur une extension des MDP avec des contraintes sur les récompenses ; leur méthode de résolution, reposant sur un ensemble d'équations de programmation dynamique avec des paramètres, permettait alors de trouver une solution sous la forme d'une politique aléatoire.

Ce résultat de politique aléatoire est perturbant en ce qu'il est difficilement applicable à un cadre industriel : cela revient à effectuer un jet aléatoire avant une décision, pour que sur plusieurs trajectoires les contraintes soient respectées en moyenne. Bien que mathématiquement correct, il est complexe de convaincre des autorités de validation qu'il est nécessaire d'introduire de l'aléatoire dans la décision. D'autres travaux se sont donc par la suite concentrés sur les CMDP sous hypothèse de la recherche d'une politique déterministe. C'est par exemple le cas de Chen et Feinberg [Che80], qui ont montré qu'il était possible sous cette hypothèse d'effectuer une résolution par programmation dynamique.

Denardo et al. [FR12] se sont intéressés à la conversion d'une politique aléatoire en politiques déterministes ; ils ont montré que cette conversion pouvait être effectuée en se basant sur la mesure d'occupation, et ont pu appliquer cette conversion aux CMDP. Cette approche a pour avantage de présenter une politique solution (c'est-à-dire une stratégie qui répond à notre problème de planification) qui paraît plus sensée à un opérateur humain : en présentant plusieurs politiques déterministes, on présente en réalité plusieurs procédures différentes, même si le choix entre ces procédures devrait demeurer aléatoire.

4. Linear Programming

Parmi les modèles s'inspirant du cadre CMDP, on pourra retenir une extension de CMDP aux contraintes sur ressources continues, développées par Meuleau et al. [MBB+14] : ces travaux proposent un algorithme heuristic inspiré de AO* pour la planification stochastique avec contraintes, appliquée à la planification de robots d'exploration autonomes. Dans cette application, les contraintes sont autant une garantie de sécurité qu'une assistance lors de la résolution, dans les cas ou ces contraintes permettent de limiter l'espace d'états qu'il est nécessaire d'explorer.

II.3.2 Modification de la fonction de récompense

Une manière naturelle d'intégrer des contraintes aux MDP consiste à modifier la fonction de récompense : la fonction de récompense, associant à chaque action une valeur positive ou négative, guide naturellement la recherche de solutions vers certaines zones tout en évitant d'autres zones. Cette classe de méthode est cependant disjointe des CMDP, puisqu'ils n'apportent pas les même garanties en termes de respect des contraintes : si toutes les solutions violent les contraintes, un planificateur CMDP retournera une erreur, alors qu'un planificateur MDP (avec fonction de récompense modifiée) retournera une solution, même si cette solution a potentiellement une valeur anormale. Pour distinguer les deux garanties apportées, on parlera de **contraintes dures** lorsque l'algorithme assure par construction que les contraintes sont vérifiées ; on parlera de **contraintes faibles** lorsqu'une vérification supplémentaire doit être effectuée après la génération de la solution pour vérifier quelles contraintes sont respectées. Notons que, sous certaines hypothèses sur les valeurs maximales des récompenses ainsi que le type de critère d'agrégation choisi (valeur moyenne, valeur dévaluée, horizon fini), il est possible de montrer que les techniques de modifications de récompense offrent des garanties fortes. Cependant, la vérification de ces hypothèses peut s'avérer tout aussi complexe que la résolution du problème. Les deux méthodes d'extensions doivent donc être perçues comme complémentaires plutôt que concurrentes.

Bacchus et al. [BBG96] ont défini une classe de MDP pour lesquels la fonction de récompense est dépendante de l'histoire du système. Cette dépendance peut par exemple être exprimée au travers d'une Logique Temporelle sur le passé, appelée PLTL. Ces formules de Logiques Temporelles permettent donc de récompenser le système dès lors que certains comportements ont été effectués, ou au contraire de le pénaliser. Bacchus et al. ont montré qu'il était possible de transformer le problème en un problème MDP avec une structure de récompense classique.

Le même principe a servi d'inspiration aux travaux de Gretton et al. [GPT04] puis de Thiébaux et al. [TGS+06] : ces auteurs se sont intéressés à des cadres MDP à récompenses non-markoviennes (NMRDPP), en s'appuyant sur des Logiques Temporelles LTL, respectivement PLTP et FLTL pour le passé et le futur. Les planificateurs sont alors capables de convertir le problème en un MDP classique, à travers l'ajout de variables temporelles. On notera cependant que le cadre NMRDPP qu'ils ont défini est plus général que celui des fonctions de récompenses définies à partir de Logiques Temporelles. Ces travaux sont intéressants pour notre problème, en ce qu'ils effectuent un rapprochement entre décision et Model-Checking, tout en présentant des algorithmes qui demeurent efficaces en temps de calcul. Cependant, ils présentent deux restrictions : ils ne permettent pas d'exprimer des contraintes dures et nécessitent donc une vérification postérieure ; et ils ne permettent pas d'exprimer des contraintes probabilistes, c'est-à-dire des contraintes qu'il faut respecter avec au moins une certaine probabilité. Plus précisément, l'aspect probabiliste des contraintes est mélangé avec l'optimisation, puisqu'une contrainte à respecter est transformée en un bonus positif ou négatif sur un chemin. La proportion de respect d'une contrainte se retrouve donc transformée en une proportion de ce bonus à prendre ou éviter. Nous voyons donc que ces deux restrictions découlent au final de la même restriction, qui est que les contraintes utilisées sont des contraintes faibles. Ce constat sera potentiellement limitant pour les cas d'application que nous envisageons, puisque nous souhaitons obtenir une garantie forte sur le respect des contraintes.

D'autres travaux se sont à l'inverse concentrés sur la validation de contraintes probabilistes. Courcoubetis et Yannakakis [CY90] ont par exemple étudié la synthèse de contrôleur valide selon le respect de propriétés exprimées sous la forme d'un automate (via un langage ω-régulier). La résolution est effectuée en cherchant à optimiser la probabilité de réalisation de ces propriétés, ce qui se traduit en une forme particulière de MDP qu'il est possible de résoudre. À nouveau, les propriétés à vérifier sont transformées en récompense, mais avec pour objectif de maximiser la réalisation des propriétés.

Sans modification, ce modèle ne permet pas de réaliser l'optimisation d'un paramètre sous certaines contraintes : il privilégiera dans tous les cas la validation des contraintes, même si la solution ne sera alors pas optimale.

Les travaux de Teichteil [TK12b] sur les Stochastic Safest and Shortest Path (S3P) problems contrebalancent en partie ce défaut : ils proposent un critère à deux composantes, cherchant à optimiser à la fois une récompense ainsi que la probabilité d'atteindre un but. On ne parle en revanche plus de contraintes, puisqu'il n'y a pas de comparaison finale entre la probabilité obtenue et un seuil fixé par l'utilisateur. À nouveau, la conclusion vis-à-vis de notre problème est que l'utilisation de structures de récompenses particulières pour exprimer des contraintes ne semble pas apporter de garanties suffisantes pour que l'on puisse considérer qu'il s'agisse de contraintes dures.

II.3.3 Synthèse de contrôleur valide

Des résultats inverses sont obtenus lorsqu'on se concentre uniquement sur la recherche d'une solution validant les contraintes, en délaissant la composante d'optimisation de récompense : les travaux de **Bacchus** et al. [BBG97] en 1997 sont particulièrement intéressants, en ce qu'ils exploitent la même logique PLTL du passé pour obtenir un résultat différent. Ils ont défini le cadre NMDP (Non-Markovian Decision Process) dans lequel cette Logique Temporelle permet d'exprimer des dépendances entre états, à partir d'opérateurs tels que "since", "always in the past", "once" ou "previously". Ils ont alors prouvé qu'il était possible de convertir les problèmes de cette classe en MDP structuré par l'ajout de variables temporelles. La résolution de ce MDP nous permet d'obtenir une politique valide, dont tous les chemins possibles respectent les contraintes PLTL.

De manière similaire, Baier et al. [BGL$^+$04] ont réalisé des études sur la synthèse de contrôleur PCTL : l'objectif est de trouver une politique, pour laquelle l'exécution respecte des contraintes PCTL. Ils ont prouvé qu'une telle politique devait être dépendante de l'histoire du système et aléatoire. Ils ont de plus montré que ce problème de synthèse de contrôleur était d'un niveau de complexité supérieur au problème de résolution MDP : il s'agit d'un problème NP-difficile. Ce résultat est essentiel, puisqu'il montre mathématiquement la différence entre MDP et synthèse de contrôleur valide : s'il était possible de réduire le problème de synthèse de contrôleur à un processus décisionnel markovien, ces deux problèmes auraient la même complexité. Ce résultat est d'autant plus difficile à percevoir que les exemples les plus faciles de contraintes PCTL peuvent souvent être ramenés à des contraintes sur des états à éviter ou à atteindre avec une certaine probabilité, ce qui peut être résolu grâce aux méthodes des paragraphes précédents.

Enfin, les travaux de Younes et Simmons [YS04] sur le planificateur TEMPASTIC ont permis d'obtenir des résultats similaires pour la logique CSL, se basant sur du temps continu. Pour s'attaquer à la complexité du problème, ils ont procédé en deux étapes : en premier lieu une réduction du problème à un problème de planification déterministe, puis la construction de la politique sous la forme de prévision de réparations du plan.

Ces trois résultats ont pour limitation principale de ne pas permettre d'optimisation en parallèle de la validation : les solutions trouvées vérifient les contraintes, mais si plusieurs solutions étaient valides ces algorithmes ne pourraient pas garantir de trouver la meilleure.

II.3.4 MDP avec objectifs multiples

Enfin, des travaux ont été réalisés plus récemment sur le rapprochement des processus décisionnels markoviens et de la validation de contraintes exprimées en Logique Temporelle. Etessami et al. [EKVY07] ont ainsi proposé un cadre de MDP multi-objectifs, pour lequel plusieurs propriétés LTL sont exprimées et doivent être validées avec une probabilité minimale fixée. Bien que ces travaux ne présentent pas d'optimisation d'une fonction de récompense cumulée, ils sont intéressants en ce que les solutions trouvées sont obtenues en construisant un front de Pareto, montrant la relation de compensation qui existe entre les différentes contraintes : augmenter la probabilité de respecter une contrainte peut par exemple avoir pour effet de baisser la probabilité d'une autre ; les courbes de Pareto permettent d'exprimer ce type de relation. Comme pour les travaux précédents sur ce même thème, les solutions trouvées sont alors des politiques aléatoires avec mémoire.

Ce type de modèle a pour avantage de pouvoir être facilement étendu à d'autres types d'objectifs, y compris des objectifs d'optimisation de récompense. Les travaux de Forejt et al. [FKN+11] [FKP12], réalisés dans le cadre du planificateur PRISM, proposent aussi un cadre multi-objectifs pour les MDP avec une résolution grâce au front de Pareto. Les objectifs permis sont plus larges : il s'agit d'objectifs qui doivent être atteints avec une certaine probabilité minimale, d'objectifs de fonction de récompense à maintenir dans une certaine enveloppe ou d'objectifs de maximisation d'autres fonctions de récompenses. La solution trouvée est constituée de plusieurs politiques déterministes, ainsi qu'un point sur chacune de ces politiques représentant un jet aléatoire qui doit être effectué dans l'état initial. Parmi les modèles que nous avons détaillés jusqu'ici, ces travaux sont les plus proches des modèles qui seraient nécessaires pour la résolution de notre problème, en ce qu'ils présentent à la fois une composante de contrainte forte et une composante d'optimisation. En termes d'expressivité du modèle, il sera nécessaire de regarder plus en amont dans quelle mesure nous pouvons représenter notre problème dans ce formalisme. Le défaut principal est cependant celui de la performance en termes de temps de calcul : la résolution repose, dans un sens, sur plusieurs résolutions MDP puis sur la résolution d'un système d'équations linéaires pour trouver les poids entre les différentes politiques déterministes solutions. Ce processus n'est pas applicable en l'état pour des problèmes de grande taille, comme les résultats des tests applicatifs de Forejt et al. ont pu le montrer.

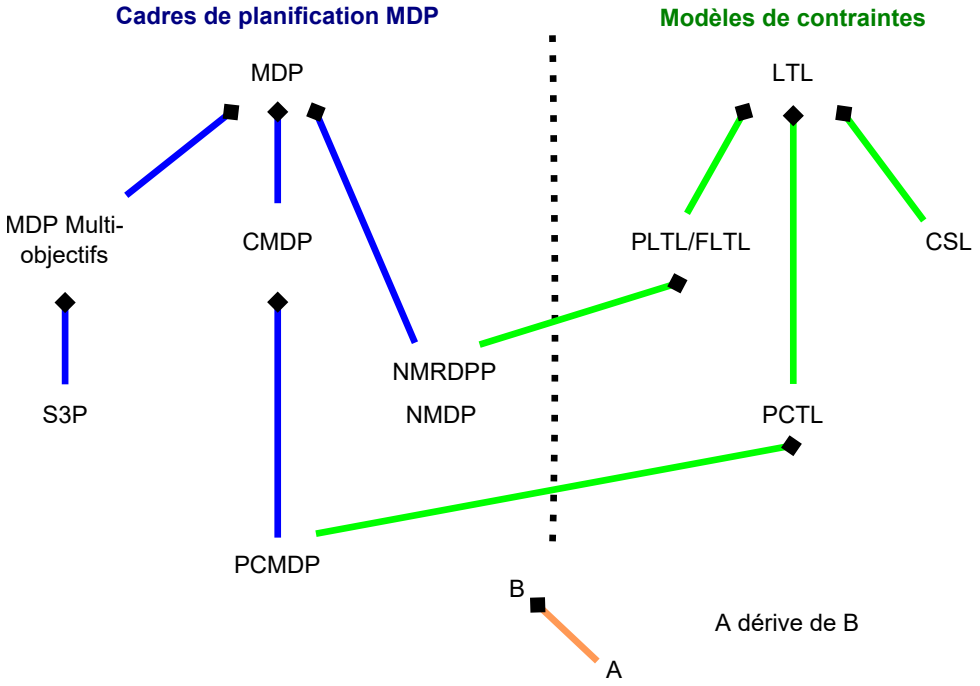


FIGURE 9 – État de l'art de la conception de système sûre et optimale, centré sur le rapprochement entre planification MDP et contraintes de Logique Temporelle.

Les travaux de Teichteil [TK12a] obtiennent des résultats similaires, par une méthode différente : Teichteil y définit le cadre PCMDP, consistant à résoudre un problème MDP sous contraintes de Logique Temporelle PCTL. La solution obtenue est donc valide au sens des contraintes et optimale aux sens de la somme cumulée des valeurs de la fonction de récompense. La méthode de résolution repose sur la mesure d'occupation ainsi que sur la résolution d'un problème de Programmation Linéaire. La solution obtenue est une politique aléatoire, mais non dépendante du temps puisque Teichteil impose la vérification d'une propriété de "transience" sur le modèle, c'est-à-dire de non-retour en arrière sur certains sous-ensembles d'états. En l'absence de test sur un modèle commun, il est difficile de dire si cette méthode obtient des résultats plus rapidement que la méthode précédente ; la méthode de résolution reposant sur une résolution par programmation linéaire, nous pouvons cependant en déduire qu'elle ne sera elle non plus pas adaptable à des modèles de grande taille. Nous pouvons néanmoins noter que les solutions trouvées diffèrent entre ces deux méthodes : les solutions trouvées par les travaux de Teichteil sont des politiques aléatoires imposant une valeur de dévaluation fixée par l'algorithme $\gamma < 1$, alors que les travaux de Forejt et al. sont des politiques aléatoires seulement à l'état initial et

imposant de plus un horizon fini ou une hypothèse de convergence finie de la somme des récompenses (non-dévaluées) d'une politique solution.

II.3.5 Questions ouvertes en amont de notre étude

En conclusion de cet état de l'art sur les méthodes de résolution de problèmes de synthèse de contrôleur sûre et optimale (Figure 9 page 42), on notera que les travaux rapprochant décision dans l'incertain et vérification formelle sont récents et peu représentés, relativement aux travaux sur la planification dans l'incertain et sur le Model-Checking. Ceci est en partie dû au changement de classe de complexité : la plupart des modèles développés dans le cadre d'un tel rapprochement ont une classe de complexité NP-difficile ou pire, ce qui limite d'autant leur applicabilité sur des modèles de taille industrielle.

Cependant, des études significatives ont été réalisées dans ce sens, prouvant la faisabilité du problème : il est possible, bien qu'avec des techniques peu efficaces, de résoudre des problèmes de conception sûre et optimale selon une approche "Décider en Vérifiant". Les questions qu'il nous faut adresser dans cette étude, qui s'inscrivent dans la continuité des résultats obtenus par le passé, sont donc les suivantes :

- Est-il possible de concevoir un algorithme capable de résoudre le problème de conception sûre et optimale sur des modèles de taille industrielle ?
- Quelles sont les hypothèses sur la forme des contraintes et la forme de la décision qui sont les plus adaptées ?
- Peut-on adapter les langages de capture de connaissances (STRIPS, PPDDL...) originaires du monde de la planification dans l'incertain pour prendre en compte les contraintes venues du monde de la vérification formelle (PCTL) ?
- Ou inversement, comment adapter les langages propres au Model-Checking (Altarica, NuSMV, ...) pour prendre en compte les éléments nécessaires à la synthèse de contrôleur ?

Première partie

Étude d'un problème de décision en maintenance avionique

CHAPITRE III

REPRÉSENTATION DU PROBLÈME SUR UN CAS D'AIDE À LA DÉCISION APPLIQUÉE À LA MAINTENANCE AVIONIQUE

Sommaire

L'OBJECTIF principal de nos travaux est de concevoir un processus outillé assistant la conception d'un système critique. Comme nous l'avons évoqué précédemment (Définition 12 page 15), un processus outillé fait référence à un ensemble de méthodes de travail et de méthodes de représentations des connaissances, dont la mise en œuvre s'appuie sur des outils qui ont été adaptés spécifiquement à ce processus.

Au cœur de ce processus outillé reposent plusieurs modèles mathématiques permettant de capturer des connaissances, puis de manipuler cette connaissance pour concevoir un système respectant les critères de qualité de service et les contraintes de maîtrise des risques (Définition 13 page 16).

La construction de ce processus outillé passe donc par les étapes (non-ordonnées) suivantes :

- Le choix du ou des modèles mathématiques permettant de manipuler la connaissance.
- La définition des modèles mathématiques nécessaires à la capture de la connaissance.
- La définition des formats d'expression des résultats obtenus à chaque étape du processus.
- L'étude des traductions et communications entre ces différents modèles.

En nous intéressant au problème de conception sûre et optimale, sous une approche "Décider en Vérifiant", nous nous intéressons en premier lieu aux changements apportés à la première de ces étapes, c'est-à-dire le choix des modèles mathématiques de manipulation de la connaissance. Nous parlons de plusieurs modèles, puisque le processus complet est potentiellement constitué de plusieurs étapes, chacune contribuant à remplir des critères et des objectifs de natures différentes, où d'étapes impliquant des technologies différentes en fonction du problème à traiter. L'objet de cette étude est donc en premier lieu de déterminer un périmètre de modèles mathématiques pouvant être mis en œuvre dans un processus de conception sûre et optimale, et si nécessaire de combler les lacunes dans ce périmètre à l'aide de nouveaux modèles ou de nouveaux ponts effectuant la liaison entre les modèles existants.

Pour déterminer de quoi serait constitué un processus outillé de conception sûre et optimale, en termes de modèles mathématiques, il nous faut donc établir quelles sont les étapes successives de la conception d'un système critique. Pour cela, nous allons analyser un cas d'application industriel, à partir d'un scénario informel, dans le but de déterminer quel modèle de la littérature serait le mieux à même de représenter le problème de conception sûre et optimale pour chacune des étapes mises en œuvre.

Ceci nécessite notamment d'identifier des caractéristiques précises sur les données du problème, en particulier sur la dynamique du système à concevoir : on souhaite savoir si le système est contrôlable ou non, probabiliste ou non, à temps discret ou continu. À partir de ces caractéristiques, si un tel modèle n'existe pas, nous souhaitons au travers de ce cas d'application être en mesure de proposer un nouveau modèle.

III.1 Étude d'un scénario d'aide à la maintenance avionique d'un business jet

L'avionique désigne l'ensemble des systèmes informatiques et électroniques présents dans un avion, et par extension dans un aéronef. Ce terme désigne donc à la fois les composants logiciels (c'est-à-dire les différents programmes informatiques permettant le contrôle de l'avion) ainsi que les composants matériels (c'est-à-dire les ordinateurs embarqués, les cartes au sein de ces ordinateurs qui sont dédiées à des tâches précises, les câbles de communication et d'alimentation, ainsi que les écrans et les commandes de vols qui sont face aux pilotes dans les cockpits).

Dans un avion moderne, on trouve notamment des équipements et logiciels de radiocommunication, de systèmes de navigation (VOR, GPS,...) , de pilote automatique, ainsi que des équipements dédiés à des fonctions plus haut niveau telles que des assistances à l'atterrissage (ILS).

La caractéristique qui nous intéresse plus particulièrement dans le domaine avionique est sa criticité (Définition 9 page 12) : chacun des sous-systèmes d'une suite avionique doit être soumis à une analyse de risque, montrant qu'à la fois le matériel et le logiciel n'induisent pas un risque inacceptable. Une telle analyse est aussi menée lors de l'intégration des sous-systèmes, pour vérifier que des risques ne sont pas introduits à l'interaction entre les équipements. Le domaine général où sont réalisées de telles analyses est celui de la sûreté de fonctionnement [Vil88] :

Définition 20 (*sûreté de fonctionnement*)

On parle de ***sûreté de fonctionnement*** *pour désigner l'aptitude d'une entité à satisfaire à une ou plusieurs fonctions requises dans des conditions données. La sûreté de fonctionnement est considérée comme la science des défaillances et des pannes.*

Note sur la sûreté de fonctionnement : Il est important de noter qu'on parle en revanche de **sûreté** [1] pour désigner l'insensibilité du système aux virus informatiques ou aux actes malveillants tels que des tentatives d'intrusion.

Chacun de ces deux domaines (sûreté et sûreté de fonctionnement) possède ses propres critères définissant ce qu'est un risque et ce qu'est un risque acceptable ou non. Dans la suite de notre étude, nous nous limiterons principalement à la sûreté de fonctionnement, c'est-à-dire à l'étude de la criticité d'un système vis à vis de sa résistance aux défaillances.

Traditionnellement [Vil88], la sûreté de fonctionnement est évaluée selon quatre composantes :

Définition 21 (*Critères d'évaluation de la sûreté de fonctionnement*)

- *La* ***fiabilité****, c'est-à-dire la capacité du système à fonctionner correctement durant un temps donné. Elle est souvent mesurée en termes de taux de défaillance, décrivant la probabilité que le système ne tombe pas en panne avant un instant t donné.*
- *La* ***maintenabilité****, c'est-à-dire la capacité du système à être remis en état de fonctionnement. Elle est souvent mesurée en termes de temps moyen d'intervention.*
- *La* ***disponibilité****, c'est-à-dire l'aptitude du système à accomplir une fonction requise à un instant donné. On parlera par exemple de la disponibilité de la communication GPS. Il est possible de la mesurer en probabilité que le système ne soit pas défaillant à un instant donné.*
- *La* ***sécurité****, c'est-à-dire la capacité du système à ne pas conduire à des accidents inacceptables.*

Dans ce contexte, concevoir des systèmes critiques revient donc à concevoir des systèmes en procédant à une analyse de risques à plusieurs étapes de leur conception. Cependant, cette criticité a des impacts dépassant le cadre de la conception d'une suite avionique, puisqu'elle impacte aussi l'ensemble des logiciels et services qui entourent le système durant sa durée de vie. C'est par exemple le cas de la maintenance des systèmes, qui doit elle aussi respecter des règles précises veillant à assurer et maintenir la sécurité du système.

Dans la suite de cette section, nous allons présenter les enjeux et méthodes propres à la maintenance des systèmes avioniques, en les envisageant sous l'angle de la criticité : en étudiant comment les processus

1. "security" en anglais

de maintenance actuels permettent de garantir ou non un niveau satisfaisant de maîtrise des risques et de qualité de service, nous pourrons définir quels critères seraient nécessaires pour concevoir de nouveaux systèmes de maintenance automatisés ou partiellement automatisés.

III.1.1 Description du domaine de la maintenance avionique

La maintenance est une partie essentielle de la vie d'un avion, en particulier sur le plan financier et contractuel : pour un avion, tout retard ou empêchement au décollage représente des pertes financières conséquentes, à la fois pour la compagnie aérienne et pour l'avionneur (le fabricant de l'avion) ou les équipementiers (les fabricants des équipements constituant l'avion) lorsque le retard est d'origine technique.

Les opérations effectuées lors de la maintenance d'un avion peuvent être courtes, telles que la désactivation d'un composant ou bien la vérification du fonctionnement d'un équipement, ou longues telles que par exemple le remplacement d'un équipement ou le chargement d'une base de données informatique. Cependant, la maintenance agit souvent en temps contraint, puisque ces opérations sont effectuées en général entre deux vols, sur une fenêtre d'environ 30 minutes. Pour les opérations plus longues, il est nécessaire d'enlever l'équipement fautif et de le remplacer, tandis que l'équipement en panne est réparé en atelier spécialisé. On parle de **maintenance en ligne** [2] lorsque les opérations de maintenance sont mineures et effectuées directement sur l'avion près de la porte d'embarquement, et de **maintenance en atelier** [3] lorsque les opérations de maintenance sont plus complexes et nécessitent des équipements particuliers, par exemple pour désassembler un composant ; ces deux types de maintenance correspondent à des buts différents : la maintenance en ligne remet en état l'avion, tandis que la maintenance en atelier répare un équipement.

Les systèmes embarqués dans un avion sont conçus pour pouvoir être réparés facilement : chaque système informatique est par exemple présent au sein d'un bloc qui peut être enlevé et remplacé rapidement par un bloc similaire. Ces blocs modulaires interchangeables sont appelés LRU pour **Line-Replaceable Unit** ; lorsqu'une panne doit être réparée rapidement, un opérateur de maintenance doit alors identifier quel LRU est défaillant, le réparer si possible ou demander sinon son remplacement. On parlera de **dépose** lorsque l'opérateur doit retirer le LRU et le remplacer par un composant similaire entreposé près de l'aéroport ; on parlera de **reconfiguration** lorsque l'opérateur de maintenance n'a pas besoin de fournir un composant de remplacement, voire n'a pas besoin de retirer de LRU, et peut utiliser d'autres équipements à bord pour remettre l'avion en état opérationnel.

Dépose et reconfiguration sont des solutions immédiates : elles permettent de remettre rapidement l'avion en état opérationnel, mais ne permettent pas toujours de le remettre en état nominal, c'est-à-dire tel que tous ses systèmes fonctionnent au maximum de leurs capacités. Pour un LRU déposé, il s'agira ensuite en atelier de déterminer quel sous-ensemble du LRU est défaillant : un LRU est composé de plusieurs cartes, appelées SRU pour **Shop-Replaceable Unit**, qui peuvent être remplacées par une carte équivalente une fois le LRU ouvert en atelier. Pour une reconfiguration, l'équipement défaillant est programmé pour une réparation ultérieure, lorsque l'avion ne sera plus en service.

Le choix de la réparation, dépose ou reconfiguration d'un équipement dépend de quatre facteurs :

1. Une recommandation (minimale ou détaillée) peut être effectuée par le système (au travers du monitoring, du diagnostic, ou des outils d'aide à la décision intégrés) ou par des outils extérieurs (logiciels d'assistance).
2. La criticité de la panne est évaluée à partir d'un document réglementaire : la (Master) Minimum Equipment List.
3. Les procédures de maintenance permettent le "Troubleshooting", c'est-à-dire l'amélioration des résultats du diagnostic au moyen de tests interactifs.
4. Les aspects de logistiques sont aussi déterminants, lorsque des pièces ou équipements sont nécessaires pour la remise en service.

Nous détaillons ces quatre aspects dans les paragraphes suivants.

2. Line Maintenance
3. Shop Maintenance

Monitoring, Diagnostic et Aide à la décision

La prise en charge d'une défaillance dans un avion passe par trois étapes :
- **Le Monitoring**, qui consiste à collecter des informations sur chaque système individuel à l'aide de BITEs (Built-In Test Equipments).
- **Le Diagnostic**, qui recoupe ces informations pour en extraire les causes racines.
- **L'Aide à la décision**, qui assiste l'opérateur de maintenance dans les choix de réparation, lui permettant de traiter une large quantité d'informations en un temps contraint.

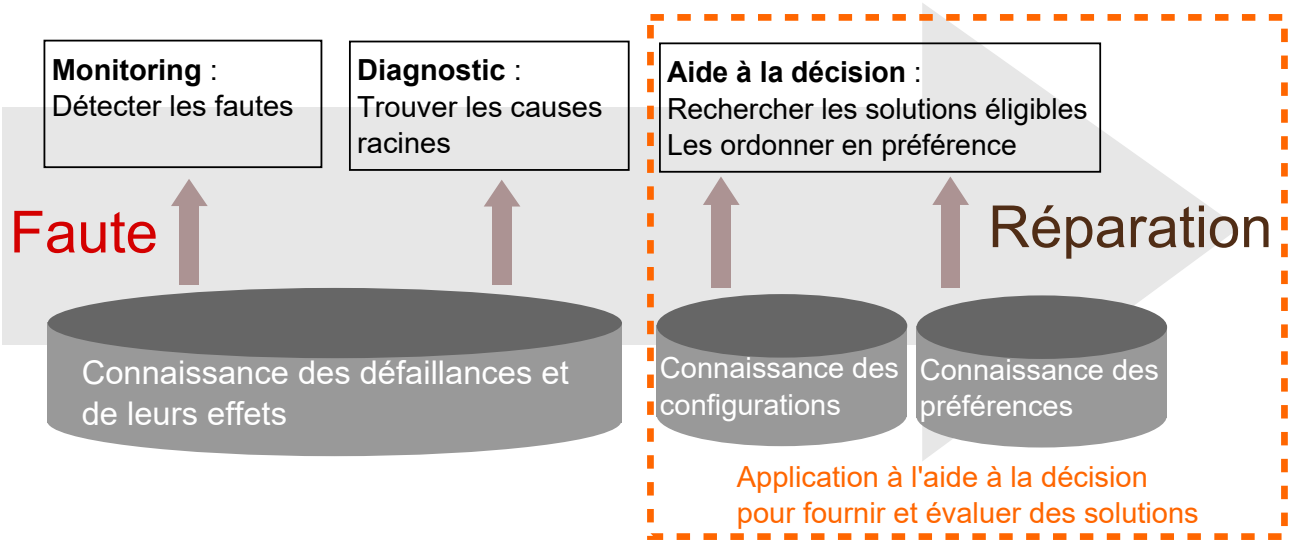


FIGURE 10 – La maintenance avionique : un processus en 3 étapes

Chacune de ces étapes (Figure 10 page 51) repose sur des technologies propres, ainsi que sur des bases de connaissances qui permettent de les alimenter. Notons que pour ces trois domaines, la plupart des recherches récentes privilégient une approche basée sur des modèles [4] [Kun13], permettant de traiter des systèmes de plus en plus complexes.

Master Minimum Equipment List

La Master Minimum Equipment List (MMEL) [EASb] est le document de référence pour évaluer si une réparation est nécessaire ou non. Elle catégorise les systèmes qui sont autorisés à être inopérants lors d'un vol : un équipement en panne est associé à un statut **No-Go** si l'avion n'est pas autorisé à décoller lorsqu'il est en panne, à un statut **Go** s'il est autorisé à décoller malgré la défaillance et à un statut **Go-If** s'il est autorisé à décoller sous certaines conditions, par exemple que l'équipement redondant ne soit pas en panne. Notons qu'on parle de Master Minimum Equipment List pour désigner le document réglementaire fourni par un avionneur sur un modèle d'avion donné et de Minimum Equipment List pour désigner le document fourni par les compagnies aériennes qui étend la MMEL - par exemple en choisissant de ne pas autoriser l'avion à décoller dans certaines conditions que la MMEL autorise.

Quel que soit le statut d'un équipement, la MMEL indique de plus un temps maximal de réparation, appelé **Intervalle de réparation** [5] après lequel cet équipement doit impérativement être réparé. Ce temps est généralement exprimé sous la forme d'une lettre A (temps spécifié dans les remarques), B (3 jours), C (10 jours) ou D (120 jours).

Une des notions importantes associée à la MMEL est celle de **Dispatch Reliability**, définie comme le pourcentage de vols qui partent sans provoquer d'interruption de service. Améliorer ce Dispatch est l'un des enjeux principaux des études portant sur la maintenance : on évalue par exemple qu'au delà de 15 minutes de retard, une compagnie aérienne perd en moyenne 72 Euros par minute supplémentaire [CTA04].

4. Model-Based
5. Repair Interval

Du point de vue de la conception des systèmes, réduire ce coût revient par exemple à concevoir des systèmes plus facilement réparables, à choisir des architectures donnant une MMEL moins contraignante, à améliorer la diagnosticabilité d'un système pour rendre le diagnostic plus précis et rapide, ou encore à prévoir des outils pour assister l'opérateur de maintenance prenant en compte les aspects logistiques.

Notons qu'on parlera d'**Operational Reliability** pour désigner le pourcentage de vols qui partent sans provoquer d'interruption d'origine technique (par opposition au Dispatch Reliability qui considère toutes les origines).

Procédures d'isolation et réparation

Pour réparer un équipement défaillant, l'opérateur de maintenance se base sur des procédures décrites dans le **Fault Isolation Manual** (FIM), appelé aussi **Trouble-Shooting Manual (TSM)**. Ces procédures sont en trois étapes :

- **Fault Confirmation**, qui indique les tests (visuels, via une interface appropriée ou un autre équipement) à effectuer pour vérifier que la panne est bien présente.
- **Fault Isolation**, indiquant une succession de matériels physiques à tester, et les manipulations pour les corriger s'ils sont défaillants (telles que le redémarrage, le chargement de fichiers de configuration, le remplacement d'une pièce,...).
- **Return To Services Tests**, indiquant les tests à effectuer pour vérifier que la panne a bien été corrigée.

La plupart des architectures modernes prévoient des équipements spécifiques pour la maintenance : un **système de maintenance centralisé (CMS)** [6] désigne l'équipement effectuant le diagnostic du système, en collectant et corrélant les différents messages de surveillance, tandis que le terme **OMS (On-board Maintenance System)** désigne l'ensemble des Built-In-Test Equipments, ainsi que les calculateurs dédiés à la collecte de messages de surveillance et l'interface de communication avec l'opérateur de maintenance - c'est-à-dire qu'il porte l'ensemble des équipements nécessaires au processus présenté dans le schéma (Figure 10 page 51). Cette interface permet alors de présenter une vue synthétique de l'état de l'appareil et d'interroger de manière interactive le système pour effectuer des tests en profondeur. Ces équipements sont présents sous une certaine forme dans la plupart des avions modernes, depuis le Boeing 777 [Ram92].

Planification et logistique

On distingue deux types de maintenance : le premier type est la **Maintenance programmée** ou maintenance préventive, où l'opérateur de maintenance intervient conformément à un planning, indépendamment de l'état de l'avion et des informations disponibles. Ces tâches de maintenance sont réalisées à une fréquence fixe appelée **Interval Check**, qui est exprimée en heures de vol sous la forme de lettre A,B,C et D.

L'autre type de maintenance est la **Maintenance non programmée** ou maintenance corrective ou curative, où l'opérateur intervient sur une panne majeure, en fonction des conditions MMEL associées. Si possible, ces réparations sont effectuées durant les temps où l'avion n'est pas en opération, par exemple la nuit pour les avions en Turn Around Time (TAT) qui effectuent une boucle sur le même parcours plusieurs fois durant la journée.

Toutes ces opérations de maintenance nécessitent des moyens particuliers, tels que du personnel qualifié, des équipements spécifiques pour réparer les pannes les plus délicates, ainsi que des pièces de rechange. La plupart des compagnies aériennes organisent ainsi un réseau de distribution de pièces de rechange et de personnels dans des escales stratégiques. De manière plus générale, un **Maintenance Control Center** (MCC) est chargé d'organiser les aspects de planification et logistique : il s'agit d'une station au sol recevant en temps réel les informations de maintenance de plusieurs avions et pouvant proposer et planifier des solutions de réparation.

6. Centralized Maintenance System

III.1.2 Maintenance : résumé des problématiques

Comme nous l'avons vu, l'objectif principal des recherches sur la maintenance avionique est d'augmenter le taux de Dispatch Reliability. Ceci peut être fait de deux manières :

- *Lors de la conception du système*, en concevant les systèmes de manière à faciliter les opérations de maintenance.
- *Lors des opérations*, en améliorant l'infrastructure et la prise de décisions des différents acteurs pour minimiser le temps ou le coût que représente chaque maintenance.

Problématiques lors de la conception du système

Le premier point peut lui-même être adressé de plusieurs manières :

- Améliorer le système global pour diminuer le nombre et l'impact des défaillances. Ceci peut par exemple être effectué en augmentant la fiabilité des équipements ou leur capacité de reconfiguration.
- Améliorer les systèmes dédiés à la maintenance, de manière à pouvoir obtenir précisément et rapidement l'identité du composant à réparer. Ceci peut être obtenu en améliorant les capteurs, les modèles de détection et le diagnostic.
- Améliorer l'interaction avec les opérateurs de maintenance ; ceci peut être obtenu en concevant des procédures et entraînements adaptés, des logiciels d'assistance ainsi que les possibilités de retour d'expérience.

Monitoring et Diagnostic ont fait l'objet de recherches communes pour leur permettre de traiter des systèmes embarqués réalistes. Les recherches récentes ont montré en particulier qu'une approche basée modèle permettait d'obtenir une meilleure qualité du diagnostic, une meilleure couverture du système en termes de diagnosticabilité [Kun13], ainsi qu'une meilleure intégration globale avec le processus de maintenance [Ram92] [BKDD04].

L'amélioration de la fiabilité des équipements est spécifique à chaque équipement ; nous ne l'aborderons donc pas dans cette étude. Nous pourrions cependant aborder les modifications d'architecture qu'il est possible d'effectuer pour augmenter la fiabilité totale du système. Puisque ce résultat nécessiterait d'étudier dans quelle mesure plusieurs équipements ou logiciels peuvent se compenser dans le cas de la perte d'un équipement, nous nous ramenons finalement à un problème de reconfiguration.

Enfin, l'amélioration de l'interaction avec les opérateurs de maintenance a été peu étudiée à notre connaissance. Ceci est en partie le résultat du processus menant à la conception des messages d'alertes et de maintenance : dans les avions actuels, ces messages sont conçus pour être porteur de l'information essentielle, permettant directement à un opérateur de démarrer les tâches nécessaires pour la réparation. Ils sont conçus au travers de spécifications fournies par un avionneur, elles-mêmes résultant d'une étude de maîtrise des risques.

L'historique du domaine y joue par ailleurs un rôle important : un opérateur de maintenance habitué à travailler sur un avion A320 aura plus de facilité à réaliser des tâches sur un nouvel avion s'il retrouve un certain degré de similarité dans le système ou dans les messages envoyés.

La pratique a cependant montré que ces messages de maintenance étaient loin d'être suffisants. Des outils réalisés par des entreprises externes proposent des aides à la décision pour les opérateurs de maintenance, assistant à la réalisation de procédures de diagnostic actif, de procédures de réparation, voire à la compréhension des messages. La conception de ces outils semble cependant entièrement détachée de la conception des équipements : ils sont réalisés bien après que les systèmes embarqués aient été conçus, en se basant sur les spécifications fournies par l'avionneur.

Problématiques de niveau opérationnel

Concernant le second point, c'est-à-dire les améliorations possibles au niveau opérationnel, les pistes suivantes peuvent être envisagées :

- Améliorer la disponibilité des équipements de rechange et du personnel de maintenance.

- Optimiser les opérations de maintenance qui sont prévisibles pour réduire le coût, par exemple en combinant des réparations sur des équipements difficiles d'accès.
- Optimiser les opérations de maintenance en fonction du temps d'escale disponible, ainsi que du plan de vol à venir.
- Améliorer les logiques de maintenance conditionnelle.
- Améliorer la maintenance préventive pour changer un équipement uniquement lorsque cela est nécessaire, mais tout en s'assurant que cela ne causerait pas de retard au décollage par la suite.

La satisfaction de contraintes logistiques est un domaine largement étudié, en particulier dans d'autres domaines que l'avionique. Il a été montré qu'une modélisation des décisions d'une compagnie aérienne concernant la maintenance [ZSZM10] permettait d'améliorer la disponibilité au niveau de la flotte, tout en permettant d'identifier des points d'améliorations à destination des équipes de design.

Les outils d'Aide à la Décision actuels se concentrent en général sur une seule approche très spécifique, telle que du pronostic. De nombreux travaux choisissent une approche statistique, par exemple pour la maintenance conditionnelle [LN09] ; certains choisissent une approche de modélisation physique ou de données [HZTM09], à travers l'utilisation de techniques telles que les Hidden Markov Models.

Ces deux aspects (contraintes logistiques et pronostic) se combinent dès lors qu'on souhaite anticiper ou différer des réparations, tout en prenant en compte leur impact sur la suite de la mission. Ce problème est particulièrement intéressant dans notre cas, puisqu'il s'agit d'une prise de décision sous contraintes, dans un environnement probabiliste. Bien que logistique et pronostic soient deux domaines largement étudiés dans la littérature, à notre connaissance aucune étude existante ne permet de générer automatiquement des scénarios de réparations possibles, prenant en compte à la fois les aspects de maintenance curative (i.e. la réparation d'une panne venant de se produire) et d'anticipation (i.e. la prise en compte des futures pannes possibles).

Conclusions sur l'étude préliminaire de la Maintenance Avionique

Comme nous l'avons établi en introduction, notre objectif est de concevoir un processus outillé pour la conception sûre et optimale d'un système critique. Dans le cas de la maintenance avionique, le système critique qui est concerné est en réalité le processus de maintenance, c'est-à-dire les infrastructures, personnels (comprenant leur formation) et logiciels qui permettent la maintenance d'un modèle d'avion particulier. Il s'agit bien d'un système critique, puisque toute défaillance de ce système a un impact potentiel grave ; il s'agit bien d'un problème de conception sûre et optimale, puisque ce système peut être évalué en termes de qualité de service (niveau de Dispatch Reliability) et de maîtrise des risques (MMEL).

Nous avons établi que l'étape principale du processus de maintenance qui impacte ces critères et contraintes se situe au moment de la prise de décision de la réparation d'une panne en ligne : c'est à ce moment que se rencontrent les contraintes de MEL, les contraintes de Dispatch ainsi que les contraintes logistiques. Cette étape correspond, en termes mathématiques, à la génération de scénarios de réparations destinés aux opérateurs de maintenance, c'est-à-dire de règles guidant la décision de réparation.

Notre objectif principal est ainsi de proposer un **outil d'Aide à la Décision pour la maintenance avionique** permettant de générer de tels scénarios : partant du diagnostic d'un système, éventuellement avec une ambiguïté résiduelle, un tel outil produirait automatiquement une ou plusieurs solutions de réparation prenant en compte les contraintes logistiques (i.e. quoi réparer, où et quand), les contraintes de sécurité (Minimum Equipment List, Interval Check,...) ainsi que les pannes futures pouvant se produire (probabilité d'occurrence d'une panne, pronostic).

III.1.3 Description informelle du scénario de synthèse de plan de réparation pour un business jet

Nous avons déterminé au cours des paragraphes précédents que le problème de conception sûre et optimale trouve une application en tant qu'aide à la décision pour la maintenance avionique. La

prochaine étape est donc de déterminer quelles sont les caractéristiques de ce problème, en particulier concernant la dynamique du système : comme nous l'avons évoqué lors de l'état de l'art, les modèles de décision que nous utiliserons seront radicalement différents selon que le système subit ou non des évènements probabilistes, selon qu'il est observable ou non, ou encore selon la forme des contraintes.

Pour cela, nous allons appuyer nos réflexions sur un exemple concret : considérons la maintenance d'un avion de type Business Jet, c'est-à-dire d'un avion d'affaire, qui présente des enjeux différents des enjeux rencontrés par les avions de ligne.

Pour cette classe d'avion, il est particulièrement difficile de garantir qu'il n'y aura pas d'interruption au sol pour cause technique (i.e. annulation du Dispatch de l'avion) : le plan de vol est susceptible de subir de nombreuses modifications, et la programmation des actions de réparation doit donc être aussi souple et adaptative que possible. De plus, à l'inverse d'un avion de compagnie aérienne, qui peut bénéficier d'infrastructures et de pièces de rechange dans plusieurs escales ou se permettre de consacrer quelques heures de sa rotation journalière pour effectuer des réparations, un Business Jet doit optimiser au mieux le lieu et le temps de chaque réparation pour en réduire le coût total.

Une solution est de confier cette gestion à une tierce partie, souvent désignée par **Maintenance Control Center** (MCC), qui se charge de gérer la réservation et l'acheminement des ressources aux escales appropriées.

Une nouvelle intelligence rendue possible par de nouveaux moyens de communication

Le contexte de la maintenance de ce type d'avion a radicalement changé au cours des dernières années en raison de l'utilisation répandue de nouvelles technologies de communication : de nombreuses technologies récentes visant à améliorer la maintenance avionique se basent sur la mise en place d'un lien de communication direct entre l'avion et le sol. En utilisant un système de communication digital dédié (appelé généralement *datalink*), il est ainsi possible de transmettre en temps réel des données sur les défaillances d'un avion, permettant aux personnels de maintenance d'identifier et de planifier les actions de réparation et de maintenance alors que l'avion est toujours en vol.

Ce concept repose à la fois sur une structure adaptée à bord de l'avion (i.e. un CMS moderne et ouvert sur l'extérieur, un système de communication tel que le système ACARS [7] pour transmettre les données de maintenance au sol,...) et sur une infrastructure logicielle au sol disponible pour le MCC. Idéalement, le logiciel récupérant les pannes en temps réel permettrait aussi de recouper les informations provenant de plusieurs bases de données, telles que les stocks disponibles, les plans de vols, et les informations de sécurité (MEL, Interval Check,...).

C'est dans cette optique que nous proposons un outil d'aide à la décision adapté, permettant de gérer tous ces flux d'informations et d'assister le MCC dans la décision de réparation.

Présentation d'un exemple de scénario

Dans ce contexte, notre cas d'étude est le suivant :

1. Un Business Jet effectue un trajet Paris - Berlin - Nice - Athènes. Une panne survient en vol entre Paris et Berlin ; un message est envoyé immédiatement au MCC.
2. Le MCC évalue la criticité de la panne selon la MEL, confirme la prise en charge de la panne et commence à planifier la réparation avant même que l'avion n'ait atterri à Berlin.
3. Le MCC consulte différentes bases de données : plan de vol, Logbook (historique des pannes), statistiques sur les temps de réparation de chaque panne, occurrences de panne des autres systèmes, pièces de rechanges et personnels disponibles à chaque escale, ...
4. Le MCC utilise un solveur pour proposer des solutions de réparation : réparer à Berlin mais avec un retard supplémentaire au sol, réparer à Nice mais demander au client d'emporter une pièce de rechange depuis Berlin, ...
5. Le solveur classe ces solutions selon plusieurs critères de préférence. Le MCC sélectionne celle qui lui semble la plus appropriée.

7. Aircraft Communication Addressing and Reporting System

6. Le MCC demande confirmation au pilote, réserve les différentes ressources et génère les "Job-cards" résumant la procédure aux opérateurs de maintenance.

L'outil d'aide à la décision que nous envisageons a donc comme objectif principal de générer automatiquement des plans conditionnels menant à la réparation de la panne considérée, en prenant en compte :

- les bases de données disponibles.
- les contraintes de sécurité imposées par la législation.
- les pannes futures éventuelles pouvant changer le plan initial.

Nous parlons de *plan conditionnel* puisqu'un plan solution doit prévoir le cas où un nouveau système tomberait en panne entre Berlin et Nice, empêchant de réaliser la réparation prévue à Nice ; il doit proposer dans ce cas une autre action de réparation prenant en compte cette éventualité. En particulier, il est impératif d'interdire certains scénarios, qui pourraient mener à des situations inacceptables, comme se retrouver dans l'impossibilité de décoller à une escale (No-Go) suite à une panne imprévue qu'il est impossible de réparer ici.

Résumé des enjeux décrits dans le scénario

Comme nous l'avons décrit dans les paragraphes précédents, nous souhaitons optimiser la prise de décision dans le cadre de la maintenance avionique. La complexité du problème vient en premier lieu de la multiplicité des informations à prendre en compte, telles que les données sur les équipements, les lieux de maintenance possibles, les temps de réparation et les coûts de chaque opération. À ceci s'ajoute la présence d'une incertitude sur certains résultats, dès lors qu'on souhaite prendre en compte l'ambigüité résiduelle lors du diagnostic ou encore l'apparition de nouvelles pannes pouvant survenir, ce qui mène à un nombre potentiellement important de solutions envisageables : il nous faut à la fois déterminer où et quand réparer, mais aussi comparer différentes méthodes pour obtenir et transporter des pièces de rechange, ainsi que la possibilité de combiner ou non des réparations.

Face à une telle complexité, un opérateur humain ne pourra pas trouver intuitivement la meilleure solution ; un **outil d'aide à la décision pour la maintenance** est donc adapté pour isoler les solutions qui semblent les plus intéressantes selon les critères de l'opérateur. En termes de plus-value applicative, le développement d'un tel outil permettrait donc de :

- **Réagir plus rapidement et plus efficacement face à une flotte d'avions**, permettant le traitement d'un plus grand nombre d'avions avec moins de personnels et la prise en compte d'un grand nombre d'informations.
- **Réduire les coûts de maintenance**, en particulier concernant l'Operational Reliability.
- **Anticiper des situations problématiques** : réparer plus tôt ou mieux, pour ne pas risquer d'être dans une situation pouvant entraîner des coûts supplémentaires ou des risques.
- **Capitaliser sur des connaissances collectées**, avec l'utilisation d'une base de données historique des pannes.

Sur le scénario choisi, cette complexité est résumée par les étapes décrites dans le scénario suivant :

MCC : Scénario d'aide à la décision

1. Le pilote est averti d'une défaillance et transmet un message au MCC
2. Le MCC acquitte de la prise en charge
3. Le MCC consulte :
– La MEL
– Le plan de vol
– Le TSM
– Une base de données historique de la défaillance
– Une liste des pièces de rechange et personnels disponibles aux escales
4. Le MCC utilise un outil d'aide à la décision pour générer des scénarios à partir de ces informations
5. L'outil d'aide à la décision organise les scénarios selon des critères :
– Privilégier le temps de réparation
– Privilégier le coût des réparations
– Privilégier le maintien de la mission
6. Le MCC effectue finalement la réservation des ressources associées

Nous avons ainsi exploité une connaissance générale du domaine pour définir un scénario informel d'aide à la décision dans la maintenance avionique, c'est-à-dire une définition informelle des informations nécessaires à un utilisateur et des objectifs que l'utilisateur doit remplir. Il s'agit bien d'une vision "informelle" puisque nous n'avons pour l'instant pas décrit de quelle manière les informations peuvent être représentées. Dans la section suivante, nous allons donc définir de manière mathématique ce problème, afin de déterminer quelles hypothèses sont raisonnables ou non dans le contexte de la maintenance avionique pour des avions de type Business Jet.

III.1.4 Étude de la modélisation formelle du scénario

Préambule : justification du choix de la modélisation formelle

Comme nous l'avons décrit en introduction, l'utilisation de techniques d'Intelligence Artificielle dans des domaines tels que la maintenance avionique est bien souvent limitée par le fait que ces domaines doivent répondre à des contraintes fortes, législatives ou contractuelles. Ajouter de "l'intelligence" à un système, par le biais d'algorithmes répondant dynamiquement à une situation, est bien souvent considéré impossible ou trop risqué lorsque des considérations de certification entrent en jeu.

C'est en partant de ce constat que nous nous sommes intéressés dans un état de l'art préliminaire aux méthodes permettant de contraindre et valider l'évolution d'un système dynamiquement reconfigurable, afin de garantir qu'il respecte les critères et contraintes requis aussi bien d'un système statique que d'un opérateur humain.

Néanmoins, nous souhaitons aller plus loin qu'une simple simulation de tous les cas possibles, telle qu'on pourrait l'envisager dans une méthode du type "Décider puis Vérifier" : à partir d'une modélisation d'un problème, nous cherchons à obtenir une garantie mathématique de la validité de nos décisions. Ce type de garantie forte peut être aujourd'hui obtenue à partir des méthodes de vérification formelles (Model-Checking), qui sont en particulier utilisées lors des étapes de vérification des systèmes avioniques ; notre but en construisant notre outil d'Aide à la décision est ainsi d'appliquer les concepts et algorithmes de ces méthodes à un autre cadre que la vérification, celui de la prise de décision dans l'incertain.

Définition d'un type de modélisation adapté à la conception sûre et optimale

Nous partons ainsi d'une approche fondée sur l'utilisation de Modèles formels de représentation des connaissances sur le système pour appliquer de telles techniques. Les modèles que nous considérons

peuvent être aussi bien de niveau composant, sous-système, système, voire représentatifs du cadre opérationnel ou d'une mission. Ils présentent plusieurs catégories d'informations, pouvant être modélisées séparément ; on peut en particulier établir la séparation suivante :

Modélisation formelle d'un problème de conception sûre et optimale

- **Modèle du monde physique** : il représente l'évolution du système dans son environnement, ses lois dynamiques indépendamment de tout agent et de toute décision.
- **Modèle des actions contrôlables** : il représente les choix "intelligents" que l'on donne au système, ainsi que les impacts de ces choix sur l'environnement. La définition de ces actions est associée intimement à des objectifs qui vont guider les décisions de l'agent dans une direction ou une autre.
- **Modèle des contraintes** : il représente les exigences que nous souhaitons garantir sur notre système. Les contraintes concernent des évolutions du système qui sont possibles physiquement, mais que l'on souhaite éviter.

Notons qu'une telle séparation recouvre un vaste panel de cadres mathématiques, selon les hypothèses que nous choisissons sur chacun de ces modèles : déterministe, probabiliste, non déterministe, temps discret ou continu, actions concurrentes ou séquentielles,... Nous détaillerons dans une partie ultérieure quelles hypothèses s'appliquent à notre cas d'étude et comment s'expriment les critères à optimiser selon le cadre mathématique retenu.

En se basant sur ce scénario et ces exigences informelles, la démarche de modélisation est alors simple : dans un premier temps nous devons définir de manière formelle les données du problème, ce qui nous amènera à poser quelques approximations pour délimiter le périmètre de la résolution. Ceci nous permet dans un second temps de choisir un cadre mathématique approprié pour modéliser ce type de problème.

Formalisation des variables représentant la totalité de la dynamique du système

En nous basant sur la catégorisation des variables exprimée précédemment, nous avons donc les modèles suivants dans notre scénario :

- **Modèle du monde physique** : probabilité et effets des défaillances de chaque système, logistique et pièces disponibles, plan de vol ...
- **Modèle des actions contrôlables** : actions de réparations et d'approvisionnement en pièces de rechange,... associées à un indicateur de coût.
- **Modèle des contraintes** : MEL, Interval check, Interval repair,...

Avec quelques simplifications, nous définissons le problème du Business Jet de la manière suivante (Figure 11 page 59) : un avion est composé de N systèmes, indexés par des numéros. Chaque système S_n peut tomber en panne avec une probabilité p_{S_n} ; celle-ci est exprimée soit relativement à une durée fixe de temps, par exemple la probabilité de tomber en panne par heure de vol (Flight Hour), ou soit à partir d'un taux de défaillance λ_{S_n} si nous faisons l'hypothèse d'une loi sans mémoire (i.e. une hypothèse que la loi de défaillance du système est une loi exponentielle).

De manière plus générale, il est possible de prendre en compte un pronostic plus précis sur la défaillance d'un système en considérant $F_{S_n} : \mathbb{R} \to [0;1]$ la fonction de répartition de la loi de défaillance, telle que $F_{S_n}(t_1) - F_{S_n}(t_0)$ soit la probabilité qu'une défaillance du système S_n se produise entre les temps t_0 et t_1.

Remarquons qu'une telle fonction de répartition peut alors même dépendre d'autres paramètres, tels que la date de dernière réparation ou vérification du système, permettant d'optimiser la maintenance préventive dans notre modèle : lors d'une escale, un solveur nous indiquerait s'il est pertinent d'effectuer

une vérification plus approfondie de l'état d'un système, pour obtenir des informations plus précises sur sa probabilité de défaillance, voire un remplacement anticipé d'une pièce.

Chaque système peut être réparé en utilisant un ensemble de ressources $R^{need}_{S_n}$, qui est simplement défini comme une fonction caractérisant un sous-ensemble de toutes les ressources existantes ; cette réparation a une durée fixe t_{S_n}. Les ressources peuvent être indifféremment des pièces de rechange, du personnel qualifié ou des outils spécifiques nécessaires à la réparation.

L'avion a un plan de vol de M escales. À chaque escale E_m est associée une durée $t^{repair}_{E_m}$ où l'avion est disponible pour une réparation, une durée $t^{gate}_{E_m}$ où l'avion est au sol se préparant au décollage et où aucune réparation n'est possible, ainsi qu'une durée $t^{next}_{E_m}$ qui est la durée en vol entre l'escale E_m et E_{m+1}. Au total, l'avion reste donc au sol à l'escale E_m pendant une durée $t^{repair}_{E_m} + t^{gate}_{E_m}$, mais seulement une partie de ce temps peut être consacrée à la réparation. Chaque escale E_m possède une liste de ressources disponibles $R^{available}_{E_m}$; ces ressources sont soit déjà présentes sur le site, soit acheminées par le MCC à temps.

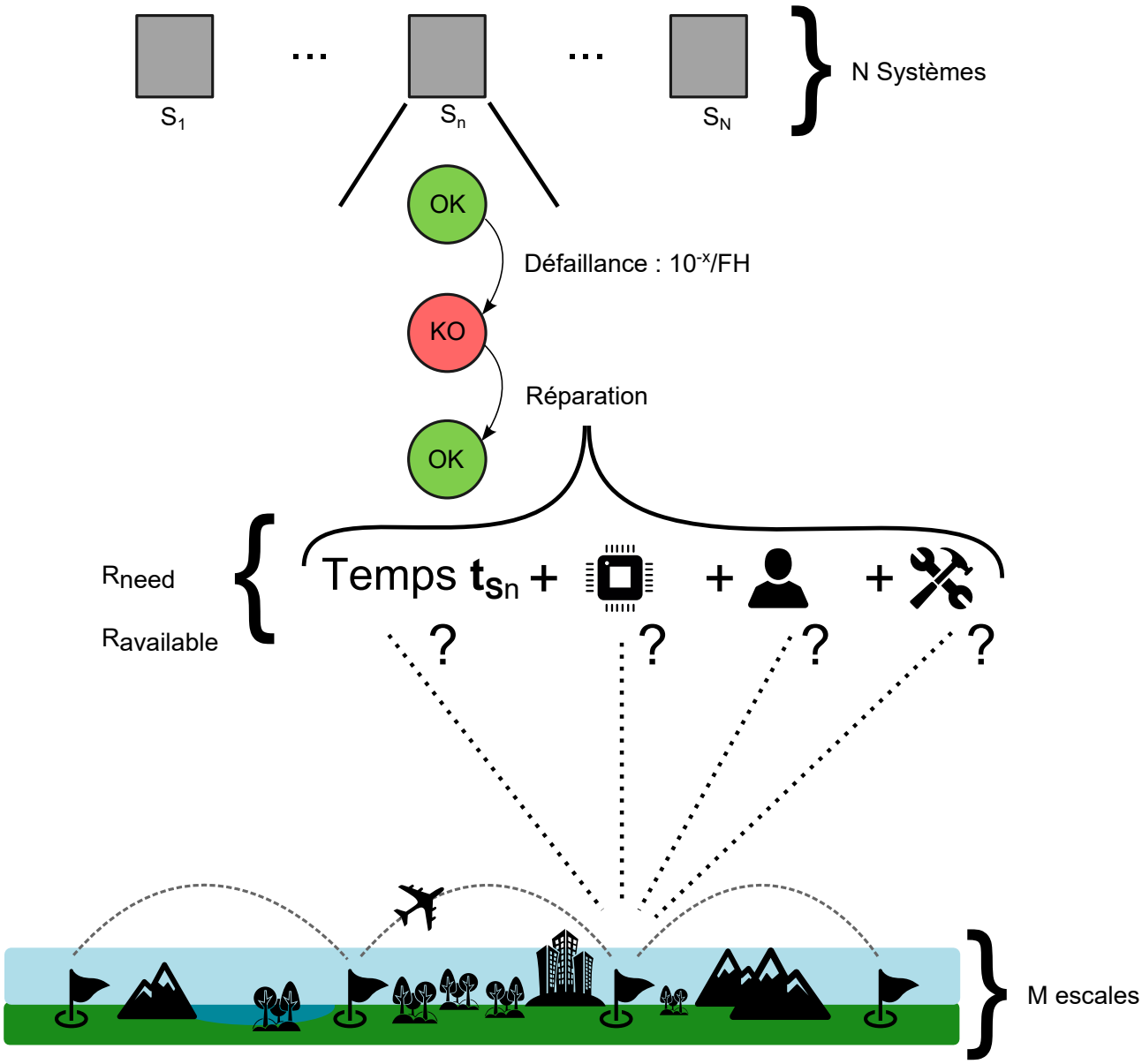


FIGURE 11 – Illustration des variables du problème du Business Jet

Formalisation des variables représentant la décision

À chacune des ressources de $R_{E_m}^{available}$ est associée un coût $c(r, E_m)$ correspondant au coût de mise à disposition de la ressource r à l'escale E_m (par exemple le coût d'achat et d'expédition d'une nouvelle pièce de rechange). Ce coût prend en compte le temps d'expédition de la pièce compte-tenu du plan de vol : il faudra par exemple choisir une expédition en "express" pour pouvoir mettre à disposition une pièce de rechange à une escale m, ce qui impactera le coût correspondant.

Les actions de réparation consistant à prendre une pièce à une escale pour l'utiliser à une autre sont pré-calculées à l'aide de $R_{E_m}^{available}$: cette pièce est considérée disponible aux escales successives, avec le coût de la mise à disposition dans l'escale de départ ajouté au coût de transport. On considère que le temps de chargement d'une pièce en soute à bord est négligeable. Si une escale E_m permet à la fois l'option de fournir une pièce directement (en expédiant sur place) ou d'utiliser une pièce emportée depuis une escale précédente, on considèrera toujours la solution ayant le moindre coût. Il est alors facile de retrouver après-coup l'origine d'une pièce de rechange, et d'effectuer les démarches pour la mettre à disposition au bon moment.

Formalisation des variables représentant les contraintes

À chaque système est associé un nombre de jours MEL_{S_n} correspondant au nombre maximal de jours où ce système est autorisé à rester en panne sans être réparé. En première approximation, nous considérons que ce nombre de jours ne dépend que du système S_n et non de l'état global de l'avion.

Ce modèle de données peut-être formalisé sous la forme d'un schéma UML minimal (Figure 12 page 62).

III.1.5 Conclusion de la capture formelle du scénario

Le point de départ de ce chapitre a été de spécifier une problématique industrielle mettant en œuvre un processus de conception sûre et optimale. Ceci a été effectué dans la première section, puisque nous avons identifié qu'un tel processus s'avèrerait nécessaire pour aider les acteurs de la maintenance avionique à faire face à la complexité de la prise de décision, en particulier face à la grande quantité d'informations qu'il est nécessaire de mettre en relation.

À partir de ces données disparates, nous avons formulé dans cette section un scénario d'aide à la décision pour la maintenance avionique qui illustre cette problématique. La définition formelle de ce scénario contribue à notre problème global de deux manières :

- elle constitue **un cas de test** (benchmark) pour lequel il est facile de générer aléatoirement de nouvelles données. Habituellement, ceci permet à la communauté scientifique d'évaluer des méthodes de résolution, selon des critères de qualité qui sont proches des critères de l'industrie (tels que l'Operational Reliability ou le coût total de réparation) et permettent donc de justifier de la valeur ajoutée de ces méthodes pour l'industrie.
- elle abstrait les **données mathématiques** de ce scénario, ce qui permet de se concentrer sur la partie de modélisation et de résolution algorithmique du processus outillé complet. En particulier, on abstrait de cette manière les problématiques de connexion à des bases de données existantes, de traduction de données sous différents formats, ainsi que les documents de traçabilité à fournir tout au long du processus.

Ces deux contributions ont été apportées successivement par une description informelle des données, donnant un contexte industriel à l'objectif qui doit être rempli, puis par une formalisation abstraite des données introduisant les variables pouvant être paramétrées ainsi que les critères d'évaluation, à savoir le coût et les contraintes de temps de réparation et de MEL.

Cette formalisation des objectifs permet cependant beaucoup de libertés : il n'est par exemple pas précisé dans cette formalisation de quelle manière les différents coûts peuvent être regroupés pour constituer une mesure de coût global, ou dans quelle mesure les contraintes de temps ou de MEL peuvent être ou non violées, avec une certaine pénalité. La raison de cette liberté est que cette formalisation

est du ressort du choix d'un modèle mathématique : certains modèles, ainsi que certains algorithmes de résolution de ces modèles, permettront des garanties différentes sur le respect de contraintes ou sur la qualité globale ; certains modèles permettront d'obtenir une seule décision, une liste de décisions, des conseils sur les décisions à suivre ou un branchement de décisions possibles... C'est ce choix de modèle mathématique précis que nous devons à présent effectuer.

Dans la section suivante, nous partirons ainsi de ce scénario formalisé pour évaluer les modèles existants dans la littérature. Cette évaluation a pour objectif d'étudier l'impact des hypothèses de ces modèles sur l'intérêt industriel du problème, et permettra de déterminer un périmètre des technologies qui sont amenées à être utilisées au sein des processus de conception sûre et optimale. À partir de ce périmètre, nous pourrons enfin évaluer si des modèles ou algorithmes de résolution sont actuellement manquants, et si nous sommes en mesure de proposer un nouveau modèle ou algorithme pour pallier ce manque.

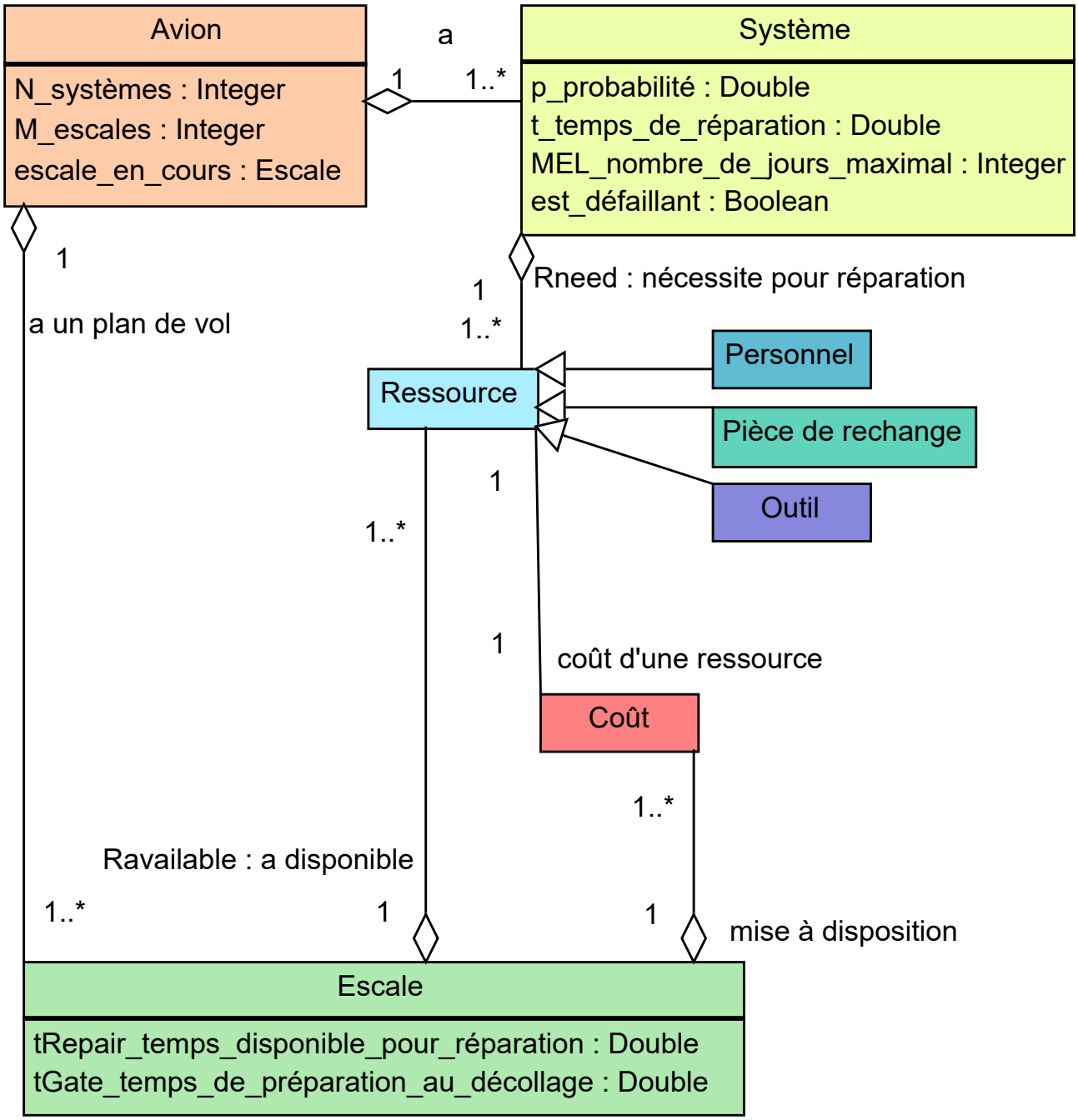


FIGURE 12 – Schéma UML de la formalisation du scénario Business Jet

III.2 Sélection d'une approche mathématique

Nous avons établi au cours des paragraphes précédents une formalisation des données du scénario d'aide à la décision pour la maintenance d'un avion de type Business Jet. Cependant, cette formalisation laisse deux questions, auxquelles il nous faut à présent répondre :

1. Dans quelle mesure les modèles existants de la littérature permettent de résoudre ce problème ?
2. Quelle est la conséquence du durcissement ou de l'assouplissement de certaines hypothèses sur la solution obtenue ?

III.2.1 Étude de l'influence des hypothèses sur le temps de décision

La première hypothèse que nous devons étudier est de savoir si nous considérons une seule décision à un instant donné, si nous considérons des décisions successives, et encore si nous pouvons prendre ces décisions successives à certains temps prédéfinis ou de façon continue.

Décision unique

Dans le cas où l'objectif est de prendre une décision de réparation unique, la prise de décision se résume à un problème d'optimisation : sachant que certaines réparations sont possibles ou non en fonction du temps et des ressources disponibles au point d'atterrissage, l'opérateur doit trouver la meilleure action permettant de remettre en service l'avion, respectant les contraintes imposées par la MEL et avec le moindre coût dans la mesure du possible.

En termes de méthodes mathématiques, ce type de problème peut par exemple être résolu par des techniques de type **CSP (Constraint Satisfaction Problem)**, lorsque le coût à optimiser doit respecter une certaine borne, ou plus généralement des techniques **d'optimisation sous contraintes**, qui permettent de trouver la valeur de certaines variables à partir d'un ensemble d'équations et d'inégalités, de façon à optimiser une somme pondérée des variables. Les données formelles que nous avons définies dans les paragraphes précédents se prêtent bien à une traduction sous la forme d'un problème de programmation linéaire à variables discrètes (Algorithme 2 page 64).

Nous pouvons de plus choisir d'assouplir certaines des contraintes lors de la résolution, en les traduisant en paramètres à optimiser. Ceci concerne par exemple :

- **Le temps de réparation**, pouvant dépasser le temps maximal alloué au sol sous une certaine pénalité représentant un coût de retard.
- **L'aéroport d'arrivée**, prenant ainsi en compte les déviations, sous la forme d'un coût supplémentaire en carburant et en retard.

Il est cependant important de noter que ces méthodes n'exploitent pas les données connues sur les probabilités de défaillance ou les données de pronostic, ou sinon de façon indirecte : il est possible d'intégrer ces données sous la forme d'une modification du coût par exemple au travers de critères tels que le temps moyen jusqu'à la prochaine faute, associé à un facteur de criticité.

Toutefois, il n'est pas facile de mettre les critères de pronostic sur le même plan de comparaison que d'autres critères, tels que des critères économiques : il est alors nécessaire d'avoir recours à d'autres méthodes que des sommes pondérées pour concevoir une fonction de coût qui représente la situation de manière correcte. Bien souvent, ces fonctions passent par des opérateurs nécessitant un certain paramétrage, qui ne peut être effectué qu'avec un retour d'expérience sur plusieurs cas d'exploitation réels.

De manière similaire, si nous ajoutons à l'hypothèse de décision unique une hypothèse d'absence de contraintes, il est possible alors de résoudre cette classe de scénario avec des techniques d'aide à la décision multi-critères [FGE05] : il s'agit dans ce type de problème de déterminer une fonction de préférence permettant de comparer deux situations à partir d'un ensemble de variables disparates ; muni de cette fonction de préférence, il est possible d'agréger plusieurs critères, nous donnant ainsi un guide sur les bonnes et mauvaises décisions.

Néanmoins, ce choix semble problématique dans la mesure où le scénario décrit un ensemble de contraintes fortes : les contraintes imposées par la MEL ne peuvent sous aucun prétexte être brisées. La solution de contournement usuelle est de représenter le non-respect de ces contraintes par un coût réellement prohibitif, dont la valeur doit être fixée arbitrairement grande. Le dimensionnement de la valeur de ce coût dépend de la méthode d'agrégation et peut être difficile à connaître avant une première résolution. Ceci implique que de telles méthodes d'aide à la décision multi-critères sans contraintes font partie de la catégories des processus "Décider puis Vérifier", puisqu'une itération est nécessaire sur certains paramètres entre la prise de décision et la garantie de validité. Notons cependant que pour ce scénario l'étape de vérification est très simple et rapide à effectuer dès lors qu'on dispose de la liste des contraintes MEL.

Enfin, le dernier point limitant concernant cette méthode de gestion de la MEL est qu'elle ne prend en compte que les contraintes MEL liées aux composants défaillants à l'instant présent, et non les contraintes MEL qui pourraient s'ajouter par la suite, à cause de nouvelles défaillances survenant avec une certaine probabilité. Il est ainsi assez aisé de construire un scénario où un avion prendrait le risque de ne pas réparer une pièce à la première escale, pour raison de coût, pour finalement se retrouver bloqué à la deuxième escale lorsqu'un équipement redondant subit aussi une panne.

Ce type de scénario est particulièrement problématique pour les moteurs d'aide à la décision, puisqu'il concerne des évènements très rares (c'est-à-dire qui ont une très faible probabilité de défaillance), mais avec des conséquences très importantes (coût d'immobilisation, rapatriement de pièces en urgence,...). Par conséquent, il est particulièrement difficile d'associer un coût à ce scénario, d'autant plus que le retour d'expérience sur ce type de situations est faible.

Algorithm 2: Traduction simplifiée du scénario du business jet en problème de programmation linéaire

Soit m une escale fixée;
Soit $a_1, \dots, a_N \in \{0; 1\}$ des actions de réparation possible ;
Soit $c_i = \sum_{r \in R_i^{need}} c(r, m)$ le coût d'application de la réparation a_i;
Soit t_i^{failed} une durée connue depuis laquelle le système i est en panne à l'arrivée à l'escale m;
Fonction à minimiser :
$f(A) = \sum_i c_i a_i$;
Contraintes :
$\sum_i t_i a_i \leq T_m^{repair}$;
$\forall i, \left(t_i^{failed} + T_m^{repair} + T_m^{gate} + T_m^{next}\right)(1 - a_i) \leq MEL_i$;
$\forall i, \forall r \in R_i^{need}, \mathbb{1}_{R_m^{available}}(r) \geq a_i$;

Décisions successives

Une décision unique correspond à une décision à un horizon immédiat : on ne considère pas les conséquences à long terme de nos actions ; en revanche, si l'on souhaite étendre notre horizon de décision, il est nécessaire de raisonner sur plusieurs "coups à l'avance", c'est-à-dire de prévoir des décisions cohérentes sur la durée. En particulier, une décision qui semble bonne pour un horizon court peut tout à fait s'avérer mauvaise sur un horizon plus long. Pour résoudre un problème de décisions successives, il ne suffit donc pas de résoudre successivement plusieurs problèmes de décisions uniques. Comme nous l'avons détaillé dans l'état de l'art préliminaire, le problème de la prise de décisions successives est essentiellement traité dans la littérature sous l'angle de la planification autonome (Figure 7 page 33).

La planification autonome est un domaine particulièrement large de l'intelligence artificielle ; elle est souvent classifiée en fonction des hypothèses simplificatrices qui sont choisies, en particulier en répondant à ces questions :

– Les actions sont-elles **déterministes ou non-déterministes** ? Pour les actions non-déterministes, est-ce que les probabilités associées sont disponibles ?

- Les variables d'état sont-elles **discrètes ou continues** ? Ont-elles un nombre fini de valeurs possibles ?
- Est-ce que l'état courant peut-être **observé en entier** et sans ambiguïté ? On parlera dans ce cas d'observation totale, par opposition à l'observation partielle.
- Combien **d'états initiaux** y a-t-il ?
- Est-ce que les actions ont une **durée** ?
- Est-ce que plusieurs actions peuvent-être prises de manière **concurrente**, ou bien une seule action est-elle possible à la fois ?
- Y a-t-il **un seul agent** ou plusieurs agents (coopératifs ou concurrents) ?

Parmi les modèles de planification, on désigne par "planification classique" le modèle avec les hypothèses les plus simples, c'est-à-dire :

- Le système a un seul état initial, unique et connu.
- Les actions n'ont pas de durée.
- Les actions sont déterministes.
- Une seule action peut être prise à la fois.
- Il y a un seul agent effectuant les actions.

Dans la mesure du possible, en choisissant un modèle on essaiera toujours de choisir des hypothèses proches des hypothèses de la planification classique, toute hypothèse plus complexe augmentant la complexité du problème - à la fois en temps de calcul et en temps de modélisation.

Pour étudier quelles hypothèses peuvent être choisies, le problème de décision peut être formulé de la manière suivante : connaissant l'état de l'avion et des ressources à l'état initial, quelles décisions de réparation pouvons-nous prendre à chaque étape du plan de vol pour permettre à l'avion de finir sa mission ?

Nous faisons donc ici l'hypothèse de limiter le temps de décision à chacune des escales. Si nous ne faisions pas cette hypothèse, cela signifierait que l'opérateur du MCC peut être amené à prendre plusieurs décisions au cours du vol, qui influeraient immédiatement sur l'avion ; ceci serait par exemple le cas si le MCC était chargé du guidage en continu de l'avion, ou s'il pouvait agir sur des reconfigurations internes tout au long du vol pour optimiser par exemple la consommation en carburant ou le trafic aérien. Dans le modèle que nous considérons, ceci n'est pas le cas : même si le MCC peut être averti avant l'atterrissage de l'avion d'une défaillance ou d'un changement de situation, les décisions qui sont prises sont uniques pour chaque escale, à savoir le lieu d'atterrissage et la liste des réparations à effectuer.

En termes de modèle, ceci veut dire qu'il n'y a aucun intérêt particulier à choisir des modèles à temps continu, puisque les temps de décision sont discrets. De la même manière, il n'y a aucun intérêt à choisir des modèles à espace continu, puisque toutes les variables considérées sont discrètes.

Parmi les restrictions que nous pouvons imposer de manière évidente, le choix de l'observabilité totale semble ici adapté, dans la mesure où l'on se base sur un diagnostic entièrement fiable. Si en revanche le diagnostic ne donne pas un résultat certain, ou tout du moins un résultat qui peut-être rendu non ambigüe au travers d'une action, alors il nous faut considérer un modèle avec observabilité partiel tel que POMDP (Partially Observable Markov Decision Process [Mon82]). Nous supposerons pour la suite que nous disposons ici d'une observabilité totale.

Nous pouvons aussi nous restreindre à un seul agent, le MCC, effectuant une seule action à la fois. Les actions étant de plus des actions de commandes de pièces, de déviation de plan de vol ou de procédures de réparation, on peut modéliser le système de façon telle que ces actions ne soient pas durative mais instantanées, dans la mesure où les contraintes temporelles sont respectées.

Enfin, l'état initial est bien évidemment unique et connu, dans la mesure où le diagnostic est fiable. Parmi les hypothèses restantes, il ne nous reste plus qu'à étudier dans quelle mesure les actions sont déterministes ou non.

Le premier modèle à considérer est celui de la **planification classique**. Dans ce modèle, les probabilités de défaillance ne sont pas prises en compte, puisque le modèle ne considère pas des futurs évènements redoutés possibles. Une solution à un problème de planification classique nous permettrait par exemple de trouver un ensemble d'actions à réaliser lors des prochaines escales pour remettre l'avion en l'état en respectant des contraintes de MEL, lorsque cette réparation nécessite une succession d'actions. Toutefois, dès lors que l'état courant ne correspond pas à ce qui était prévu, il est nécessaire de replanifier, c'est à dire de réaliser une nouvelle résolution du problème. Ce modèle apporte néanmoins une plus-value par rapport à la simple résolution de contraintes, puisqu'il permet de traiter des cas où une seule action ne suffit pas pour résoudre le problème, par exemple s'il faut emporter une pièce de rechange dans une autre escale pour effectuer la réparation. Notons cependant que la planification classique peut être traitée sous l'angle d'une résolution de contraintes [PVLI10].

Le second modèle à considérer est celui de la **planification non-déterministe**, en particulier la planification conformante [CR00]. Dans ce modèle, les probabilités de défaillance ne sont pas prises en compte, mais le modèle prend en compte le fait que de nouvelles pannes peuvent survenir. Une solution est donc un ensemble de réparations qui garantissent de mener l'avion jusqu'à la fin de la mission, quelle que soit la combinaison de défaillances pouvant survenir. Cependant, ce type de modèle a une application limitée dans notre cas, puisqu'il existe vraisemblablement toujours une combinaison de défaillances de systèmes pouvant empêcher l'avion de mener à bien sa mission, aussi improbable soit cette combinaison.

Le troisième modèle que nous pouvons considérer est celui des processus décisionnels markoviens (Figure 7 page 33), ou plus généralement celui de la **planification en environnement probabiliste**. Une solution d'un tel modèle est un plan conditionnel, appelé politique, donnant l'ensemble des actions à effectuer pour remettre l'avion dans un état correct, tout en anticipant les scénarios de défaillance possibles et en prévoyant des actions de contre-mesures pour compenser les scénarios les plus probables. Ce modèle prend ainsi en compte les probabilités de défaillance, qui sont utilisées pour calculer un coût moyen sur chacun des futurs possibles.

Sans surprises - puisque nous avons présenté ce modèle dans un chapitre précédent - les processus décisionnels markoviens semblent au premier abord les plus adaptés : le modèle MDP est le seul permettant de couvrir toutes les données utilisées. Mais ce modèle présente aussi potentiellement une complexité plus grande que ce que nous souhaiterions : la modélisation et la résolution d'un MDP est plus complexe que l'autre modèle alternatif possible, qui est la planification classique. Avant de choisir définitivement un modèle, il nous faut donc étudier ce que chaque approche peut nous apporter.

III.2.2 Étude des modèles utilisés par les outils existants

Dans les paragraphes précédents, nous avons identifié trois modèles principaux permettant de traiter le problème d'aide à la décision pour la maintenance avionique sous trois angles différents :

- **L'optimisation sous contraintes**, permettant de trouver l'action immédiate qui remet le système dans le meilleur état possible, tout en respectant un critère de coût à optimiser.
- **La planification classique**, permettant de trouver une succession d'actions remettant le système dans le meilleur état possible à la fin de ces actions.
- **La planification MDP**, permettant de trouver une succession d'actions amenant le système dans le meilleur état possible en prenant en compte les défaillances pouvant survenir lors de la suite de la mission.

Modèles d'optimisation sous contraintes

Comme nous l'avons déjà évoqué, la modélisation du problème sous la forme d'une optimisation sous contraintes ne nécessite pas de nouvelles technologies, et des librairies telles que [CO] permettent de résoudre de tels problèmes et donnent des résultats tout à fait acceptables sur des cas d'application industriels. Ceci est aussi le cas des méthodes d'aide à la décision multi-critères, pour lesquelles les outils mathématiques de résolution sont désormais largement utilisés.

Le seul apport que nous pouvons réaliser sur ces deux domaines est celui de la capture des préférences de coût, en proposant une aide à la définition du coût en fonction des données prises en compte, par exemple d'un modèle d'impact des défaillances futures. Comme cette définition des préférences doit vraisemblablement être effectuée à partir d'un retour d'expérience, voire améliorée de façon continue par une collecte de données, nous n'avons pas poursuivi cette piste plus en avant dans cette étude. On citera cependant les études de [FH10], [SS02] et [VV98] auxquelles pourra se référer un lecteur intéressé.

Modèles de planification classique

L'angle de la planification classique demande plus d'attention, en particulier concernant la prise en compte des contraintes : les coûts des équipements et opérations de réparation peuvent être traduits directement en coût d'action, de sorte qu'une solution optimisant la somme du coût des actions optimise en réalité le coût financier total de la réparation.

De la même manière, les contraintes MEL peuvent être directement traduites en états "sans-issue", c'est-à-dire en états à partir desquels aucune action n'est possible, donc a fortiori à partir desquelles il n'est pas possible de compléter la mission. Cependant, ce modèle simple ne prend pas en compte les défaillances pouvant survenir en cours de mission : pour les prendre en compte, il faudrait s'assurer que pour toute panne pouvant survenir à tout instant du reste de la mission, il existe une solution de réparation permettant à l'avion de terminer sa mission tout en respectant les contraintes MEL. Bien qu'il soit théoriquement possible de compiler cette condition sous la forme d'une contrainte, c'est-à-dire de réaliser une première résolution du problème avec un autre modèle puis d'utiliser cette solution pour marquer certaines actions comme étant autorisées ou interdites, une telle solution est complexe en comparaison de méthodes déjà existantes pour la planification en environnement probabiliste.

Puisqu'une version simplifiée du problème peut être résolue sans encombre par des techniques existantes de planification classique, par exemple au travers des librairies SHOP2 [NAI+03] ou GRAPHPLAN [BF97], nous n'avons pas poursuivi cette piste, qui ne présente qu'un intérêt limité à la fois sur le plan scientifique et sur le plan économique - à moins potentiellement d'utiliser un cadre de planification/replanification lorsque l'état courant diffère de l'état prédit, tels que ceux existant dans la littérature [Ste95].

Modèles de planification en environnement probabiliste

L'angle de la planification en environnement probabiliste présente en revanche un intérêt clair : c'est le seul parmi les trois modèles à prendre en compte les données de probabilité de défaillance - qui sont par exemple disponibles au travers des processus d'analyse des risques lors de la conception de l'avion , tels que le processus de PSSA (Preliminary System Safety Assessment) [ARP 4761]. Il se pose en revanche deux questions pour ce modèle :

- Quel critère d'agrégation choisir pour le calcul du coût global d'une solution ?
- Comment assurer la validité de la solution choisie ?

Les trois critères de coût les plus classiques pour les modèles MDP sont les critères de somme dévaluée, de somme non dévaluée et de coût moyen. Le critère de somme dévaluée représente le fait qu'une action nous donnant une récompense immédiate a plus d'utilité (au sens économique) pour l'agent qu'une action pouvant rapporter une grande récompense plus tard ; à l'inverse, un critère de somme non-dévaluée correspond simplement à la somme du coût total de chaque futur possible, pondérée par la probabilité d'occurrence de ce futur. En première analyse, il semble que le critère de somme non-dévaluée soit le plus pertinent, puisque le MCC n'a pas un intérêt particulier à éviter des coûts immédiats, préférant minimiser le coût total ; les deux options sont néanmoins envisageables, avec une difficulté ajoutée pour la solution de critère de somme dévaluée : pour étudier cette solution, il faudra fixer un paramètre de dévaluation γ, qu'il serait vraisemblablement nécessaire d'évaluer par retour d'expérience. Le troisième critère, de coût moyen, ne semble pas correspondre à notre problème.

Notons que notre problème présente l'avantage de baser les valeurs de coût sur une réalité économique ; le fait d'additionner des coûts a donc un sens réel, contrairement à certains problèmes pour lesquels réaliser la somme de deux paramètres n'a pas de sens, voire permet de douter de la validité du résultat.

En revanche, ce constat est rendu caduc dès lors qu'on souhaite inclure la gestion des contraintes sous la forme d'une récompense fortement négative : pour interdire certaines solutions d'un MDP, il est d'usage d'interdire certaines actions en modifiant l'espace d'actions, ou de les décourager en associant des récompenses négatives à certains états violant les contraintes fixées [PCCT13]. Le fait qu'il n'y ait pas de sens (économique) à cette somme de coûts réels et de coûts artificiels implique bien souvent qu'une solution d'un tel MDP n'a aucune garantie de respecter les contraintes ; il est donc nécessaire d'effectuer une étape de vérification supplémentaire, c'est-à-dire que le processus suive un schéma "décider puis vérifier".

Comme nous l'avons vu lors de l'état de l'art préliminaire, il existe des modèles tels que le modèle PCMDP [TK12a] permettant de réaliser une approche "décider en vérifiant". Il est intéressant de noter que ce modèle particulier peut être résolu en se basant sur une méthode de programmation linéaire, c'est-à-dire qu'il transforme le modèle PCMDP en un problème d'optimisation sous contrainte. Le point positif de ces modèles est donc de montrer que l'approche MDP est faisable pour notre problème ; mais il montre par ailleurs qu'il est d'un niveau de complexité potentiellement plus grand que les deux autres modèles.

Bilan sur la comparaison des modèles existants

Le bilan de cette étude des modèles existants est donc que le choix d'une dynamique non-probabiliste simplifie suffisamment notre modèle pour qu'il puisse être résolu par des outils existants, avec l'inconvénient que nous n'exploitons pas toutes les informations dont nous pouvons disposer, en particulier puisque des replanifications sont nécessaires ; à l'inverse, le choix d'une dynamique probabiliste implique qu'il est nécessaire de considérer des modèles parmi les plus récents dans la littérature - et donc les moins matures.

Dans la suite de cette étude, nous considérerons donc l'hypothèse la plus générale, celle d'une dynamique probabiliste. En particulier, nous étudierons de manière qualitative dans quelle mesure cette hypothèse de dynamique probabiliste apporte un gain avéré en termes de coût et en termes de sécurité, ce qui sera à mettre en comparaison avec la complexité ajoutée en termes de modélisation et de temps de résolution.

III.2.3 Étude des conséquences de l'hypothèse d'une dynamique probabiliste

Nous avons établi que le choix d'une dynamique probabiliste oriente nécessairement notre choix vers le cadre de la planification en environnement probabiliste. Comme nous avons pu le détailler dans l'état de l'art préliminaire (Figure 7 page 33), ce domaine est largement dominé par des modèles basés sur les Processus Décisionnels Markoviens (MDP).

Afin d'établir si le modèle MDP est suffisant et adapté pour représenter notre problème, nous devons étudier à présent la conversion des données de notre problème, représentées selon notre modélisation formelle précédente, en des données de type MDP.

Construction d'un processus décisionnel markovien à partir des données formelles du problème du Business Jet

Dans le cas du problème du Business Jet, un certain nombre de ces paramètres peut être défini naturellement. L'espace d'états $S \subset \mathbb{R}^N \times \mathbb{R}^N \times [1; M]$ peut être défini comme l'ensemble des états $s = (t_{lastRepair}, t_{lastFailure}, m)$ tels que :

- $t_{lastRepair} : [1; N] \to \mathbb{R}$ est pour chaque système le temps depuis la dernière réparation.
- $t_{lastFailure} : [1; N] \to \mathbb{R}$ est pour chaque système le temps depuis la dernière panne non réparée.
- m est l'escale courante.

Puisque nous considérons que nous prenons la décision d'effectuer ou non des réparations à l'atterrissage de l'avion, cet espace est en réalité discret puisque les temps ne peuvent prendre qu'un nombre fini de valeurs.

L'espace d'actions $A = 2^N$ est alors l'ensemble des réparations qu'il est possible d'effectuer aux escales : $(a : [1; N] \to \{0; 1\}) \in A$ est le fait de réparer ou non chacun des systèmes. $R : S \times A \to \mathbb{R}$, la fonction de récompense, est définie par

$$R(\{t_{lastRepair}, t_{lastFailure}, m\}, a) = \sum_{n \in systeme} a(n) \sum_{(ressource \in R_n^{need})} [c(ressource, m)]$$

lorsque l'action a est possible en s. Elle représente simplement le coût des réparations prévues à l'escale courante m. Notons qu'il est possible d'enrichir la fonction de récompense avec d'autres paramètres, comme par exemple une pénalité pour les retards au décollage.

Définir la fonction de transition est en revanche plus complexe : il faut considérer le résultat global des réparations, la probabilité qu'une panne se produise durant le vol vers la prochaine escale, tout en mettant à jour les différents temps (dernière réparation et dernière panne). Bien qu'il soit possible de définir directement une telle fonction de transition, notre retour d'expérience est que l'exercice est fastidieux, n'est pas maintenable et peu robuste aux erreurs.

Comme nous le verrons, nous avons choisi deux solutions pour pallier ce problème :

- *(option 1)* exprimer ce problème en premier lieu dans un langage de modélisation, Probabilistic Planning Domain Definition Language [YS04] (PPDDL), qui peut alors être traduit immédiatement en Processus de Décision Markovien, ou
- *(option 2)* exprimer nativement ce problème dans le langage utilisé par le solveur, C++, et le connecter à des bases de données pour générer le problème. Nous discuterons par la suite les avantages et inconvénients de ces deux approches.

Traitement des contraintes dans le cadre MDP

Trouver une solution au problème du Business Jet consiste à trouver un plan conditionnel qui associe un ensemble d'actions de réparations à effectuer à chaque état possible. Une solution est dite optimale si elle optimise l'espérance du coût cumulé des réparations.

Néanmoins l'optimalité n'est pas suffisante dans notre cas puisque nous devons aussi garantir deux types de contraintes :

1. Les conditions MEL_n doivent être respectées, et donc aucune réparation anticipée ou tardive ne doit interdire à terme le Dispatch.
2. Certaines actions de maintenance programmées doivent être effectuées avant une certaine date limite.

Une solution est dite valide si le plan conditionnel permet toujours de respecter ces deux contraintes, quelles que soient les pannes pouvant survenir. Nous sommes donc en présence d'un Processus décisionnel Markovien sous contraintes : nous cherchons la meilleure solution possible, mais uniquement parmi celles qui sont valides.

De nombreux cadres mathématiques existent pour exprimer ce problème, nécessitant parfois de poser des hypothèses supplémentaires, notamment sur l'absence de boucle dans le plan de vol - ou tout du moins sur l'absence de boucle dans la solution choisie : il serait par exemple possible de le formuler en tant qu'un Goal-Oriented MDP tel que GSSP [KMW12], puisque nous pourrions alors garantir que le modèle ne présente pas de boucles de gain infini ; dans un tel modèle, des états ou chemins sont naturellement interdits s'ils ne permettent pas de finir le plan de vol, par exemple si l'avion reste immobilisé au sol à cause des restrictions dues à la MEL.

Il serait même possible de le modéliser directement en MDP, mais la démarche serait complexe : il faudrait fixer précisément un gain obtenu en finissant le plan de vol (ou en arrivant à une étape), ainsi qu'un gain "pénalité" pour interdire ou forcer certains états. L'ajustement de ces valeurs de récompense serait néanmoins artificiel et susceptible de varier au moindre changement du modèle : pour un avion

de type business jet, il est difficile de trouver un sens monétaire au fait de finir le plan de vol, et encore plus difficile de trouver une valeur monétaire pouvant être additionnée ou comparée au coût des réparations. Ce type de considérations est notamment traité dans les travaux de Kolobov et al. [KMW12].

De tels modèles ne sont néanmoins garantis valides qu'en l'absence de boucle avec gains infinis (i.e. plan de vol linéaire, pas d'action inefficace) ; si l'on souhaite considérer tous les types de modèles, tels que par exemple un avion en "turn around time" effectuant régulièrement le même plan de vol, nous devons passer à des cadres mathématiques plus généraux, en particulier des cadres prenant en compte des contraintes de chemin.

En nous basant sur l'état de l'art préliminaire (Figure 9 page 42), nous pouvons identifier plusieurs possibilités pour l'expression des contraintes, en particulier les logiques CSL [BKH99], PCTL et CTL ; la logique CSL (Continuous Stochastic Logic) a pour particularité de pouvoir gérer des systèmes à temps continu, mais les résultats disponibles dans l'état de l'art montrent que dans la plupart des cas les temps de résolution pour les problèmes de vérification CSL sont bien trop élevés pour notre besoin applicatif, et que le problème de synthèse de contrôleur valide pour des contraintes CSL n'a pas, à notre connaissance, de solution applicable à l'échelle industrielle. Notre modèle étant réduit à un système à temps continu en raison du nombre fini d'escales, nous pensons qu'il est raisonnable de ne pas choisir la logique CSL pour notre problème.

Puisque la logique PCTL est plus générale que la logique CTL, nous avons choisi dans un premier temps d'exprimer ce problème sous la forme d'un Path-Constrained Markov Decision Process [TK12a] et d'utiliser un solveur approprié. PCMDP est un cadre mathématique alliant planification en environnement probabiliste [Put94] et contraintes de Logique Temporelle (LTL, PCTL,...) [HJ94] : la solution d'un PCMDP est un plan conditionnel qui est garanti optimal et valide, c'est-à-dire un plan qui mènera sur le long terme au meilleur coût financier tout en étant sûr de respecter les contraintes dans toutes les conditions.

III.2.4 Évaluation de la taille du problème du business jet

Nous avons dans les paragraphes précédents justifié deux choix, dont chacun impose une complexité supplémentaire au problème :

- Le choix d'une dynamique probabiliste, permettant de prendre en compte toutes les données disponibles.
- Le choix de la représentation en contraintes de chemin, de type PCTL, nous permettant de garantir et de montrer facilement des preuves de garanties sur la sécurité de la solution.

En particulier, il est possible de déterminer un ordre de grandeur des performances attendues en termes de temps de résolution pour notre problème, en évaluant la taille des données d'entrée.

Une instance de notre problème est convertie sous la forme d'un MDP avec un espace d'états inclus dans $\mathbb{R}^N \times \mathbb{R}^N \times [1; M]$ et un espace d'actions de taille 2^N. Cependant, nous avons déjà évoqué dans un paragraphe précédent qu'il était possible de modifier le problème en un problème à espace discret, dès lors que l'on choisit une discrétisation adaptée pour le temps. Cette discrétisation peut être effectuée dans la mesure où tous les temps entre escales sont connus : plutôt que de choisir de discrétiser le temps selon un pas de temps fixe, par exemple en nombre de jours, nous pouvons choisir d'énumérer toutes les durées possibles, ce qui correspond aux temps entre escales et à toutes les sommes des temps entre escales consécutives. Si on nomme T_{max} le nombre de temps différents dans une telle énumération, on obtient ainsi un espace d'états ayant une taille $T_{max}^{2N}M$. Cette taille augmente donc de façon exponentielle en fonction du nombre N de systèmes pouvant tomber en panne, et augmente linéairement en fonction du nombre d'escales (en dehors de la valeur de T_{max}).

Ceci nous permet d'obtenir un dimensionnement des capacités souhaitées pour l'algorithme de résolution de ce PCMDP : en fixant des ordres de grandeur à m, T_{max} et N pour un cas d'application industriel, nous pouvons évaluer la taille totale du problème, nous permettant d'évaluer le nombre d'opérations nécessaires par un algorithme de résolution.

Pour un système industriel, le nombre d'escales considéré correspond à un nombre d'escales que l'avion envisage de parcourir avant d'arriver dans un lieu où une vérification complète pourra être effectuée ; le paramètre m a donc vraisemblablement un ordre de grandeur de 10.

Le paramètre T_{max} est majoré par toutes les sommes possibles de temps d'escales consécutives : on peut construire l'énumération des durées possibles en prenant les $m-1$ temps entre escales, puis les $m-2$ temps qui proviennent de la somme de deux temps entre escales consécutives, ... Pour $m-1$ intervalles T_{max} est donc majoré à $\frac{(m-1)(m-2)}{2}$, soit un ordre de grandeur de m^2 donc de 100. Notons qu'on obtient bien ainsi un nombre majorant, puisque plusieurs sommes peuvent être égales.

Le paramètre N correspond au nombre de systèmes critiques que l'on souhaite considérer. Sa valeur minimale correspond au nombre de systèmes impactant immédiatement la défaillance venant de survenir, par exemple les systèmes de la chaîne redondante. Sa valeur maximale correspond à l'ensemble des systèmes pouvant présenter un risque de sécurité durant la fenêtre prévue par le plan de vol, c'est-à-dire l'ensemble des systèmes dont une défaillance pourrait causer un No-Go au décollage. Dans le premier cas, l'ordre de grandeur est de 10 tandis que dans le second il est de l'ordre de grandeur du nombre de systèmes décrits dans la MEL, soit 100.

L'ordre de grandeur total du problème industriel peut-être estimé à un espace d'états de $100^{10} * 10$ soit 10^{21} **pour une version simplifiée** du problème et $100^{100} * 10$ soit 10^{201} **pour la version complète**. En termes de résolution, on souhaiterait que la version simplifiée puisse être résolue de façon presque instantanée, c'est-à-dire de l'ordre de la minute, permettant une réponse immédiate à la demande d'un pilote ; on souhaiterait que la version complète puisse être résolue dans une fenêtre de l'ordre de l'heure, puisqu'il s'agit de la durée de vol la plus classique pour les avions en Turn-Around-Time.

En comparant ce dimensionnement avec les résultats académiques pour le modèle PCMDP [TK12a], il devient évident que les méthodes de résolutions actuelles pour ce type d'algorithme ne permettent pas de traiter des tailles de type industriel : sur l'exemple décrit dans cet article, un temps de résolution de l'ordre de la minute correspond à une taille de grille de $60 * 60$ soit $3.6 * 10^3$ états. Cette valeur est à comparer à l'estimation de 10^{21} pour le nombre d'états de la version simplifiée du système soit, même si l'on considère une croissance linéaire du temps de résolution, un temps de résolution estimé de 3.10^{17} minutes (c'est-à-dire presque 10^6 millénaires !).

Cependant, le cadre mathématique PCMDP a ici le net désavantage d'être un cadre récent : ce faible résultat en termes de temps de résolution tient autant de la complexité du modèle que du manque de travaux existant sur des algorithmes de résolutions. Ainsi, même si ce dimensionnement montre bien la nécessité d'un algorithme de résolution plus efficace, il est pertinent d'évaluer par des expériences concrètes dans quelle mesure la complexité du modèle PCMDP se retrouve effectivement dans le scénario d'aide à la décision pour la maintenance avionique.

III.2.5 Conclusion de la sélection d'un modèle mathématique et résumé des apports sur la définition du scénario

Dans ce chapitre, notre objectif était de déterminer un ensemble de caractéristiques fondamentales sur le processus outillé de décision sûre et optimale. En nous basant sur un scénario informel d'aide à la maintenance avionique, nous avons pu réaliser les apports suivants :

- Nous avons dans un premier temps établi que la problématique d'aide à la décision dans la maintenance, en particulier de sélection d'un plan de réparation, était un **problème de conception sûre et optimale**. Ceci résulte notamment du fait que la décision de réparation doit respecter des contraintes de sécurité (MEL) et des critères d'optimisation de coût.
- Nous avons alors **formalisé les données du scénario**, définissant ainsi un nouvel ensemble de cas de tests pouvant être abordé selon différents angles de modélisation.
- Parmi les approches de modélisation possibles, nous avons identifié **trois catégories d'hypothèses** correspondant à trois types de modélisation : l'optimisation sous contraintes, la planification classique, la planification en environnement probabiliste.
- Nous avons montré que les outils de résolution de **l'optimisation sous contraintes et de la planification classique** permettent d'obtenir des résultats immédiats, mais qu'ils nécessitent une étude approfondie de la modélisation des coûts pour obtenir des résultats prenant en compte

toutes les données disponibles.
- Nous avons réalisé la formalisation du problème sous la forme d'un **Processus Décisionnel Markovien**, ce qui a mis en avant la nécessité de choisir un cadre de MDP prenant en compte des contraintes fortes. Nous avons sélectionné le modèle PCMDP comme pouvant représenter le problème du Business Jet.
- Nous avons établi au travers d'une **analyse de complexité** que ce modèle ne permet cependant pas de traiter le cas du Business Jet en raison de la taille du modèle.

Dans les chapitres suivants, nous allons ainsi réaliser l'étude de ce scénario au travers de la modélisation en PCMDP en deux étapes : dans un premier temps, nous proposerons un ensemble d'hypothèses simplificatrices pour PCMDP permettant de réduire la complexité du modèle. Dans un second temps, nous appliquerons ces méthodes au scénario d'aide à la décision pour la maintenance, en construisant le processus outillé global.

CHAPITRE IV

DÉVELOPPEMENT D'UN MODÈLE ET D'UN ALGORITHME PERMETTANT DE TRAITER LE CAS DU BUSINESS JET

Sommaire

Dans le chapitre précédent, nous avons étudié un cas d'application possible pour le problème de synthèse de contrôleur sûre et optimale. Le processus outillé envisagé pour résoudre ce problème repose sur un algorithme de résolution d'un modèle théorique constitué d'un Processus Décisionnel Markovien (MDP) et d'un ensemble de contraintes de Logique Temporelle de type PCTL. Une étude préliminaire de la littérature existante a montré qu'aucun modèle mathématique n'était exploitable pour résoudre ce problème théorique : la plupart des modèles se concentrent soit sur les aspects MDP soit sur les aspects PCTL, et les modèles prenant en compte les deux aspects ne sont pas assez rapides en termes de temps de calcul.

Il nous faut donc créer ou modifier un modèle existant pour combler ce manque : puisque le modèle PCMDP (Path-Constrained MDP) a été identifié comme le modèle mathématique le plus à même de représenter notre problème de façon complète, nous allons à présent nous intéresser au développement d'un algorithme de résolution de ce modèle qui soit plus efficace que l'algorithme original, éventuellement à l'aide d'hypothèses supplémentaires que nous pourrons poser sur notre problème.

Développer un nouvel algorithme de résolution implique d'en prouver la validité. Dans ce chapitre, dont les résultats principaux ont été publiés à la conférence internationale AAAI-14, nous reprenons la définition du modèle PCMDP puis définissons une version simplifiée à travers l'ajout d'hypothèses

supplémentaires : nous limitons les contraintes PCTL à des contraintes déterministes. Nous nous basons alors sur ce nouveau modèle, appelé **SPC MDP** pour Saturated Path-Constrained Markov Decision Process, pour obtenir un algorithme de résolution sensiblement plus rapide que les autres algorithmes existants permettant de résoudre cette classe de problèmes.

Notons que limiter les contraintes PCTL à des contraintes déterministes rapproche ces contraintes de la logique CTL [CES86], puisque nous enlevons, d'une certaine manière, la spécificité de PCTL qui est la prise en compte des probabilités. Cependant, il est difficile de dire si le sous-ensemble de CTL que nous créons ainsi est plus proche de PCTL, CTL voire LTL : CTL, en particulier, est défini à partir d'une syntaxe précise alternant des opérateurs "Always" et "Exists" avec des opérateurs "Finally", "Next", "Until" et "Globally". Nous choisissons pour cette raison de présenter le modèle SPC MDP comme basé sur une restriction de PCMDP, plutôt que de le présenter comme l'alliance des MDP et d'un sous-ensemble de CTL, ou d'un sous-ensemble de LTL ; ceci est en particulier cohérent avec la méthode de résolution que nous choisissons d'appliquer, qui est basée sur des méthodes de calcul de validité pour PCTL comme nous le verrons dans les sections suivantes.

IV.1 Définition du modèle SPC MDP

Tout comme le modèle PCMDP, le modèle SPC MDP est construit à partir de deux composantes : les Processus Décisionnels Markoviens et la Logique Temporelle PCTL.

Processus Décisionnels Markoviens

Comme rappelé dans l'état de l'art, les Processus Décisionnels Markoviens (MDP) [Put94] sont l'un des modèles les plus populaires pour résoudre des problèmes d'optimisation du comportement de systèmes dynamiques dans un environnement probabiliste. En particulier, nous avons rappelé précédemment (Définition 17 page 24) la définition des Processus Décisionnels Markoviens à horizon infini et récompense dévaluée (Infinite-Horizon Discounted-Reward Markov Decision Processes soit IHDR MDP), qui sont une des formes les plus utilisées de MDP : un IHDR MDP est un ensemble $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \mathcal{I}, \gamma \rangle$, où $\mathcal{S}$ est un ensemble d'états possibles, $\mathcal{A}$ un ensemble d'actions, $\mathcal{R}(s,a)$ une fonction de récompense donnant une récompense pour chaque action a accomplie dans un état s, $\mathcal{T}(s,a,s')$ une fonction de transition donnant la probabilité d'atteindre l'état s' lorsqu'on effectue l'action a dans l'état s, $\mathcal{I} : S \to [0;1]$ est une distribution de probabilité sur les états initiaux, et $\gamma \in [0,1)$ est le facteur de dévaluation.

Notons que dans la plupart des exemples de la littérature, il n'y a qu'un seul état initial, que nous noterons s_0. Dans le cas plus général, nous considérons cependant que plusieurs états initiaux sont possibles ; le système a alors une certaine probabilité fixe d'être dans chacun de ces états initiaux : lors de la simulation de ce système, on commencera par effectuer un jet aléatoire pour déterminer l'état initial effectif.

On peut remarquer de plus qu'il est toujours possible de ramener un ensemble d'états initiaux I à un état initial unique s_0, sans coût de construction supplémentaire en termes de complexité : il suffit de construire un état s_0 avec une seule action a_0 tel que pour tout état $s' \in \mathcal{S}$ les probabilités de transitions correspondent à la distribution de probabilité de l'ensemble des états initiaux, c'est-à-dire que $\mathcal{T}(s_0, a_0, s') = \mathcal{I}(s')$. Par souci de lisibilité, on considérera donc dans la suite de cette étude que l'ensemble $\mathcal{I}$ est toujours réduit à un état initial unique s_0.

Lemme 3 (*Réduction à un état initial*)

Pour tout processus décisionnel markovien $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \mathcal{I}, \gamma \rangle$, il est possible de construire un processus décisionnel markovien $\langle \mathcal{S}_2, \mathcal{A}_2, \mathcal{R}_2, \mathcal{T}_2, \mathcal{I}_2, \gamma \rangle$ tel que $\mathcal{I}_2 = \{s_0\}$ et tel que les deux processus soient équivalents par bisimilarité.

On rappelle que le terme **bisimulation** désigne une relation binaire entre systèmes de transitions d'états, assurant lorsqu'elle existe que les systèmes se comportent de la même façon : un système simule l'autre, et réciproquement.

Pour résoudre un IHDR MDP, on recherche une politique $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$ maximisant la fonction de valeur (Définition 19 page 25) $V^\pi(s_0)$, qui correspond à l'espérance du gain total (parfois appelée "utilité") attendu en appliquant la politique π à partir de l'état initial s_0. La politique maximisant cette utilité π^* est associée à la **fonction de valeur optimale** $V^* = \max_\pi V^\pi$. Pour un état $s \in \mathcal{S}$ d'un IHDR MDP, on peut aisément montrer [Put94] que la valeur $V^*(s)$ existe, est finie, et satisfait *l'équation de Bellman* :

$$V^*(s) = \max_{a \in \mathcal{A}} \left\{ \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') V^*(s') \right\} \tag{IV.1}$$

Ce résultat permet de prouver [Put94] que tout IHDR MDP contient au moins une politique optimale qui est markovienne et déterministe, c'est-à-dire de la forme $\pi^* : S \to A$ où S ne dépend pas des états visités précédemment. Ce résultat est essentiel puisqu'il rend possible des méthodes de résolution qui sont peu coûteuses en complexité, c'est-à-dire en temps de calcul. Les méthodes de

résolution classiques de IHDR MDP telles que Value-Iteration (Algorithme 3 page 76) se concentrent sur la recherche d'une telle politique.

Algorithm 3: Value-Iteration pour IHDR MDP

```
1  Procédure Value Iteration (S, A, R, T, s_0, γ, θ);
   Entrées  : (S, A, R, T, s_0, γ) tels que définissant un IHDR MDP
              θ > 0 un seuil défini par l'utilisateur
   Sorties  : π : S → A une politique déterministe approchant la politique optimale
              V : S → ℝ la fonction de valeur associée à cette politique
   Variables: V_k une suite de fonctions de valeurs
2  ---------------------------------------------------------------------
3  Initialiser V_0 à une valeur arbitraire ∀s ∈ S;
4  Initialiser k=0;
5  repeat
6  |  k=k+1;
7  |  for s ∈ S do
8  |  |  V_k(s) = max_{a∈A} Σ_{s'∈S} T(s,a,s')(R(s,a) + γV_{k-1}(s'));
9  until ∀s|V_k(s) − V_{k-1}(s)| < θ;
10 for s ∈ S do
11 |  π(s) = arg max_{a∈A} Σ_{s'∈S} T(s,a,s')(R(s,a) + γV_k(s'));
12 Retourner π, V_k;
```

Model-Checking probabiliste

Pour exprimer les contraintes au sein du modèle SPC MDP, nous utiliserons un sous-ensemble du langage PCTL *Probabilistic Real Time Computation Tree Logic* [HJ94]. Ce choix résulte des deux résultats suivants que nous avons détaillés dans le chapitre précédent :

- Les contraintes exprimées sur l'exemple du Business Jet, en particulier les contraintes imposées par la MEL, sont des contraintes probabilistes sur des trajectoires (au sens des chaînes de Markov) futures possibles.
- Le modèle PCMDP est le modèle existant qui permet d'exprimer au mieux notre problème ; ce modèle repose sur un sous-ensemble du langage PCTL.

PCTL permet d'exprimer des formules logiques sur l'ensemble des trajectoires empruntées par une politique donnée. Comme pour d'autres Logiques Temporelles, ces formules sont construites à partir d'opérateurs logiques et de fonctions booléennes $f : \mathcal{S} \rightarrow \{true, false\}$ sur l'espace d'états (Définition 16 page 22).

Dans le langage PCTL, le seul opérateur logique est l'opérateur temporel *probabiliste "strong until"* $fU^{\leq H}_{\diamond p} g$ où $\diamond$ désigne un opérateur de comparaison ($<$,$\leq$,$=$,$\geq$, or $>$). Comme détaillé dans un chapitre précédent (Définition 15 page 21), cet opérateur vaut Vrai si pour un tirage aléatoire d'un ensemble de trajectoires obtenues en suivant une politique donnée à partir de l'état initial, la formule $fU^{\leq H} g$ est vraie avec au moins/au plus/ exactement une probabilité p. La formule en elle-même est vraie si sur une trajectoire donnée la fonction booléenne f est vraie sur les états de cette trajectoire au moins jusqu'à ce que la fonction booléenne g soit vraie. f peut demeurer vraie par la suite, mais ce n'est pas une nécessité pour que cette formule soit vraie. Cependant, g doit devenir vraie sur la trajectoire avant au maximum H étapes depuis le début de l'exécution de la politique. Lorsque l'horizon de la contrainte H est égale à l'infini, g doit impérativement devenir vraie à un certain moment de la trajectoire pour que la formule soit vraie.

Les contraintes que nous souhaitons exprimer en PCTL font partie des plus simples à écrire, par exemple :

- $(true)U^{\infty}_{=1}g$: dans chacune des trajectoires possibles, le système doit visiter à un certain moment un état où la formule g est vraie (états obligatoires).
- $(true)U^{\infty}_{=0}g$: le système ne doit jamais atteindre un état où g est vraie (états interdits).
- $(\neg f)U^{\infty}_{=0}g$: le système n'est autorisé à visiter un état où g est vrai que s'il a visité un état où f est vrai (antériorité).

Muni de ces deux outils, il est possible de définir le modèle PCMDP de la manière suivante :

Définition 22 (*Path-Constrained MDP*)

Un processus décisionnel markovien à chemin contraint (PCMDP) est un vecteur $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, s_0, \gamma, \xi \rangle$, *où :*

- $\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}$ *et* s_0 *définissent un processus décisionnel markovien à horizon infini (IHDR MDP).*
- γ *est un paramètre de dévaluation.*
- $\xi = \{\xi_1, \ldots, \xi_n\}$ *est un ensemble de contraintes PCTL basées sur l'opérateur "strong until", chacune de la forme* $f_i U^{H}_{\diamond_i p_i} g_i$, *où* $f_i : \mathcal{S} \to \{true, false\}$ *et* $g_i : \mathcal{S} \to \{true, false\}$ *sont des fonctions booléennes.*

Une politique est dite ***valide*** *pour un PCMDP si elle satisfait toutes les contraintes de ce PCMDP pour les chemins d'exécution partant de l'état initial* s_0. *Une politique* π *est une* ***solution optimale*** *d'un PCMDP si elle est valide et vérifie :*

$$\forall \pi' politique\ valide, V^{\pi}(s_0) \geq V^{\pi'}(s_0).$$

Cette définition est adaptée des travaux de Teichteil [TK12a], en reprenant les notations utilisées précédemment. Notons que cette définition repose sur un sous-ensemble de PCTL, qui permet d'exprimer des contraintes qui ne sont pas seulement des conjonctions d'opérateur Strong Until ; en particulier, le langage PCTL permet de combiner plusieurs opérateurs Strong Until : f ou g peuvent être elles-même des formules PCTL, alors que dans la définition de PCMDP elles sont des fonctions labels sur les états.

IV.1.1 Définition du modèle Saturated Path Constrained MDP

En se basant sur ces travaux existants, nous avons pu définir une nouvelle classe de problèmes, appelée Saturated Path-Constrained MDP (SPC MDP) soit "processus décisionnels markoviens sous contraintes de chemin saturées". Un problème SPC MDP est ainsi un IHDR MDP avec contraintes PCTL, tout comme le modèle PCMDP, mais avec les deux hypothèses suivantes :

1. L'horizon H est infini pour tous les opérateurs Strong Until. Ceci signifie que les contraintes doivent être satisfaites à un certain point, mais sans date limite particulière.

2. La probabilité p des opérateurs Strong Until est égale à 0 ou 1. Les contraintes sont alors dites "saturées" ou "déterministes".

Formellement, on définit ainsi la classe de problème SPC MDP de la manière suivante :

Définition 23 (*SPC MDP*)

Un processus décisionnel markovien sous contraintes de chemin saturées (SPC MDP) est un vecteur $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, s_0, \gamma, \xi \rangle$, *où :*

- $\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, s_0$, *et* γ *définissent un processus décisionnel markovien à horizon infini (IHDR MDP).*
- $\xi = \{\xi_1, \ldots, \xi_n\}$ *est un ensemble de contraintes PCTL, chacune de la forme* $f_i U^{\infty}_{=p_i} g_i$, *où* $f_i : \mathcal{S} \to \{true, false\}$ *and* $g_i : \mathcal{S} \to \{true, false\}$ *sont des fonctions booléennes et* $p_i = 0$ *ou* 1.

Une politique π *associée à un SPC MDP est dite* ***valide*** *si elle satisfait toutes les contraintes de ce SPC MDP pour les chemins d'exécutions partant de l'état initial* s_0. *Une politique* π *est une* ***solution optimale*** *d'un PCMDP si elle est valide et vérifie :*

$$\forall \pi' politique\ valide, V^{\pi}(s_0) \geq V^{\pi'}(s_0).$$

Il est important de noter que la première de ces hypothèses, portant sur l'horizon infini, ne réduit pas l'ensemble des modèles qu'il est possible de traiter sous forme d'un SPC MDP : bien que l'horizon H des contraintes d'un SPC MDP soit infini, il est possible de traiter les cas où un but doit être atteint avant une date limite finie en introduisant une variable de temps discrète pour ce but dans l'espace d'états qui spécifie la durée pour chaque action. Ce résultat est établi dans le lemme suivant :

Lemme 4 (*Réduction à un horizon infini*)

Pour tout PCMDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, s_0, \gamma, \xi \rangle$, *tel que toute contrainte* $\xi_i \in \xi$ *ait une probabilité* p_i *égale à* 1 *ou* 0, *il est possible de construire un PCMDP* $\left\langle \mathcal{S}_2, \mathcal{A}_2, \mathcal{R}_2, \mathcal{T}_2, s_{0(2)}, \gamma, \xi_2 \right\rangle$ *tel que toute contrainte* $\xi_i \in \xi_2$ *ait un horizon* H_2 *infini et tel que les deux processus soient équivalents par bisimilarité.*

Démonstration : Soit un tel PCMDP, pour chaque contrainte ξ_i à horizon H_i fini, on ajoute à l'espace d'états une variable mémoire, agissant comme un compteur décomptant de H_i à 0 depuis l'état initial. L'espace d'états devient donc : $\mathcal{S}_2 = S \times \mathbb{N} \ldots \times \mathbb{N}$, où $\mathbb{N}$ est répété autant de fois qu'il y a de contraintes à horizon fini.

$\mathcal{A}$ et $\mathcal{R}$ sont inchangés par cet ajout. On forme le nouvel état initial $s_{0(2)} = (s_0, H_0, \ldots, H_n)$, en supposant que les contraintes de 0 à n soient celles avec un horizon fini. On forme la nouvelle fonction de transition $\mathcal{T}_2$ en transformant toute transition $t(s, a, s') \in \mathcal{T}$ en transition $t(s_2, a, s'_2)$ où en notant $s_2 = (s, h_0, \ldots, h_n)$ on a $s'_2 = (s', h_0 - 1, \ldots, h_n - 1)$.

Pour chaque contrainte à horizon fini $f_i U^{H_i}_{=p_i} g_i$, on modifie la fonction g_i de la manière suivante : $\forall s_2 = (s, h_0, \ldots, h_n) \in \mathcal{S}_2, g_{i(2)}(s_2) = g_i(s) \& (h_i > 0)$.

Ceci assure que l'objectif g_i ne peut pas être atteint une fois la date limite dépassée. Les autres contraintes sont inchangées.

On peut alors montrer de manière triviale que toute trajectoire du premier PCMDP peut être transformée en une trajectoire du second PCMDP qui a la même probabilité de validation pour chacune des contraintes et la même valeur cumulée dévaluée. Inversement, toute trajectoire du second PCMDP peut être transformée en trajectoire du premier PCMDP avec les mêmes propriétés.

Par conséquent, il existe une politique optimale (aléatoire et histoire-dépendante) π_2 du second PCMDP constructible à partir de toute politique optimale π du premier PCMDP, et inversement. Ces deux résultats montrent l'existence d'une relation de bisimilarité entre les deux PCMDP. ■

Puisque tout modèle SPC MDP est un modèle PCMDP, le lemme 4 précédent est a fortiori applicable au modèle SPC MDP.

La justification du choix de cette hypothèse dans la construction du modèle SPC MDP est donc quadruple :

- Ce choix n'impacte ni l'expressivité du modèle ni la performance en termes de complexité des algorithmes.
- Ce choix simplifie la compréhension du modèle et des algorithmes en limitant le nombre de paramètres.

– Un horizon fini n'exprime pas une limite temporelle mais une limite sur le *nombre maximal d'actions* qui peuvent être exécutées avant d'atteindre le but. Ceci est différent de *la quantité de temps écoulée dans le monde réel* mis pour atteindre le but. Comme la manière d'exprimer le temps réel écoulé varie en fonction des cas d'applications, par exemple entre la seconde, l'heure ou le jour, la plupart des modèles réels n'utilisent pas le concept d'horizon fini, mais utilisent des variables telles que celles décrites ci dessus pour gérer des contraintes de date limite finie.
– Comme nous le verrons par la suite, ce choix permet de rendre les politiques optimales solutions non dépendantes du temps. Combiné à l'ajout d'une autre variable mémoire, les politiques optimales obtenues par notre algorithme ne dépendent que de l'état courant, ce qui les rends indépendantes de l'histoire du système (markoviennes). Cette indépendance est bien évidemment uniquement mathématique, puisque dans les faits la dépendance a été intégrée à l'espace d'états ; mais ce résultat mathématique permet des propriétés intéressantes, par exemple pour montrer la convergence de l'opérateur de Bellman modifié que nous utiliserons.

On peut enfin montrer l'inclusion du modèle SPC MDP vis-à-vis des autres types de MDP existants, à travers le théorème suivant :

Théorème 2

1. *La classe SPC MDP contient la classe de problème des IHDR MDP et est contenue dans la classe de problème des PCMDP.*
2. *En considérant l'ensemble des instances de problème SPC MDP pour lesquelles il existe une marge finie $M > 0$ telle que la fonction de valeur de toute politique pour tous les états est définie et majorée par M pour $\gamma = 1$, cet ensemble contient les classes GSSP [KMWG11], SSP [Ber95] et FH MDP [Put94].*

Démonstration : 1. Un modèle SPC MDP est construit à partir d'un modèle IHDR MDP ; il contient en particulier tous les modèles sans contraintes, c'est-à-dire l'ensemble des IHDR MDP. De plus, le modèle SPC MDP diffère du modèle PCMDP en ce qu'il restreint les contraintes PCTL à des contraintes déterministes.

2. Bien que la définition des SPC MDP impose la condition $\gamma < 1$, il est facile d'autoriser $\gamma = 1$ lorsque les valeurs de la fonction de récompense assurent que la fonction de valeur V^π est bien définie pour toute politique, même lorsque $\gamma = 1$. Par conséquent, tout SSP et GSSP peut être vu comme un SPC MDP pour lequel les contraintes PCTL forcent le système à atteindre à un certain moment le but de ce (G)SSP. Une telle contrainte est simplement une contrainte d'états obligatoires, comme celle exprimée dans un des exemples ci dessus : $(true)U^{\infty}_{=1}g$. Les autres inclusions découlent de l'inclusion de la classe de modèle GSSP. ■

IV.1.2 Étude de la forme des solutions

Le modèle SPC MDP est particulièrement intéressant en ce qu'il n'est pas réductible à un MDP simple : il n'existe pas de fonctions ou d'algorithmes permettant de transformer un problème SPC MDP en un problème MDP, de résoudre ce problème MDP, puis de transformer à nouveau cette solution dans l'espace du SPC MDP d'origine pour obtenir une solution du SPC MDP. Ce résultat est énoncé dans le théorème suivant, et les paragraphes suivants vont en particulier être consacrés à prouver ce théorème au travers de l'étude d'un contre-exemple :

Théorème 3 (*Non inclusion de SPC MDP dans MDP*)

La classe de problème SPC MDP n'est pas incluse dans la classe de problème MDP. A fortiori, la classe de problème PCMDP n'est pas incluse dans la classe de problème MDP.

Ce résultat est important puisqu'il nous permet d'étudier de façon univoque dans quelle mesure les algorithmes de résolution existants s'appliquent : toute classe de problème pouvant se ramener à la résolution d'un MDP est incluse strictement dans la classe SPC MDP, et ses algorithmes ne sont donc pas suffisants pour résoudre des problèmes SPC MDP.

Pour prouver le théorème (Théorème 3 page 79), nous allons montrer dans les paragraphes suivants que de nombreuses démonstrations et résultats que nous avions dans le cas de MDP ne sont plus valables ici. Nous montrerons ainsi que la réduction classique du problème MDP à la résolution de l'équation de Bellman n'est plus valide, en particulier puisqu'il n'est plus possible de se ramener aux seules politiques déterministes ; nous verrons ensuite en quoi la définition même d'optimalité peut ne pas avoir de sens telle qu'elle est définie pour des processus décisionnels markoviens.

Invalidation du résultat classique sur les politiques déterministes

L'un des principaux résultats appliqué traditionnellement dans la résolution des MDP est qu'il est possible de se réduire aux seules politiques déterministes : en effet, il est possible de prouver que pour toute politique stochastique il existe au moins une politique déterministe de valeur supérieure ou égale ; trouver une politique stochastique optimale peut donc se réduire à trouver une politique déterministe optimale. À titre de rappel, une politique stochastique est une politique où un jet aléatoire doit être effectué dans chaque état pour déterminer l'action à prendre (Définition 18 page 24).

À l'inverse, dès lors qu'on ajoute des contraintes PCTL cette propriété n'est plus valide, comme le prouve le contre-exemple suivant :

On considère un système à deux états, A l'état initial et F ; comme sur le schéma ci-après, on définit deux actions a_1 et a_2 donnant respectivement des récompenses de $+1$ et 0. On y ajoute la contrainte PCTL d'état obligatoire disant qu'on doit nécessairement arriver dans l'état F avec une probabilité 1 : $trueU^{\infty}_{=1}f$ où f est une fonction booléenne sur les états valant uniquement vrai dans l'état F. Un tel modèle défini donc un SPC MDP minimal.

On remarque qu'il n'existe qu'une seule politique déterministe valide : celle qui consiste à choisir a_2 en A. On remarque aussi que toute politique stochastique avec $\pi(A, a_2) > 0$ est valide : la probabilité à un instant t donné qu'on soit en F étant

$$\Pr(s_{t+1} = F) = \Pr(s_t = F) + \Pr(s_t = A) * \pi(A, a_2) = (1 - \pi(A, a_2)) * \Pr(s_t = F) + \pi(A, a_2)$$

donc (suite arithmético-géométrique) on obtient :

$$\Pr(s_t = F) = 1 - (1 - \pi(A, a_2))^t \rightarrow_{t \rightarrow \infty} 1$$

La seule politique stochastique non valide est celle choisissant a_1 en A. Comme toute politique ayant $\pi(A, a_1) > 0$ a une valeur $V^{\pi}_{\gamma}(A) > 0$, la politique déterministe valide (choisissant toujours a_2) n'est pas optimale, elle est même la "moins bonne" de toutes les politiques valides.

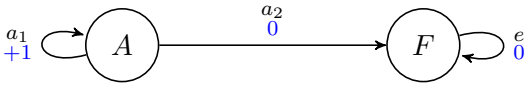


FIGURE 13 – Contre-exemple : plusieurs politiques stochastiques sont supérieures aux politiques déterministes valides

Le contre-exemple précédent nous montre bien que la réduction aux politiques déterministes n'est plus correcte ; pire, dans le cas d'un PCMDP il est possible de trouver des exemples simples où il est nécessaire de s'appuyer sur des politiques stochastiques pour trouver une solution :

On considère un système à trois états (A, G_1 et G_2) et deux actions a_1 et a_2 qui lient respectivement A à G_1 et A à G_2. G_1 et G_2 possèdent une seule action epsilon bouclant sur eux-mêmes. Toutes les récompenses sont à 0 et A est le seul état initial.

On ajoute la contrainte PCTL $\left(trueU^{\infty}_{\geq 0.5}g_1\right) \wedge \left(trueU^{\infty}_{\geq 0.5}g_2\right)$. On souhaite ainsi imposer que l'on ait une chance sur deux d'arriver en G_1 en partant de l'état initial et une chance sur deux d'arriver en G_2.

L'exemple a bien sûr été construit spécialement pour les besoins de ce contre-exemple, il semble donc évident que seule une politique stochastique pourra répondre à ces deux conditions : une politique

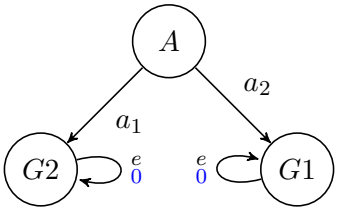


FIGURE 14 – Contre-exemple : cas où aucune politique déterministe n'est valide

déterministe arrivera soit en G_1 soit en G_2 avec une probabilité 1 et violera donc notre contrainte. Il existe en vérité une seule politique valide, celle ayant $\pi(A, a_1) = \pi(A, a_2) = 0.5$

Il existe donc des modèles pour lesquels il n'existe aucune politique déterministe valide. Ceci nous force donc, pour un problème PCMDP général, a toujours raisonner sur des politiques stochastiques. Ceci est d'autant plus difficile que l'on change d'espace (on dispose d'un spectre continu de politiques, impossibilité de prendre simplement "argmax",...) et que la littérature qui a été consacrée à l'étude de la recherche de politiques stochastiques optimales est sensiblement plus réduite.

Invalidation de l'existence d'une politique optimale

Pour montrer plus en avant les problèmes spécifiques à SPC MDP et PCMDP, reprenons le premier contre-exemple, représenté sur le schéma (Figure 13 page 80).

Cet exemple présente en vérité un autre point d'étude intéressant. Nous avons en effet montré que toute politique stochastique est valide hormis celle choisissant a_1 avec une probabilité 1 ; nous pouvons aussi calculer la fonction de valeur d'une politique avec :

$$V_\gamma^\pi(A) = \pi(A, a_1) + \gamma * \pi(A, a_1) * V_\gamma^\pi(A) = \frac{\pi(A, a_1)}{1 - \gamma * \pi(A, a_1)}$$

Cette fonction de valeur est donc strictement croissante en fonction de $\pi(A, a_1)$: plus la politique choisit l'action a_1, plus on peut s'attendre à ce qu'elle collecte une récompense élevée. Ceci implique notamment que pour toute politique stochastique valide, il existe une politique valide ayant une valeur supérieure (par exemple celle ayant $\pi'(A, a_1) = (\pi(A, a_1) + 1)/2$). La politique optimale et valide n'existe donc pas.

Cela est évidemment dû au fait que la "meilleure" des politiques ne soit pas valide : la borne supérieure de l'ensemble des politiques valides n'est pas atteinte, il est donc impossible d'en exhiber un élément particulier.

Ceci prouve donc le théorème 3, puisque ce contre-exemple montre les deux propriétés suivantes :

- *Pour un SPC MDP, il peut n'y avoir aucune politique déterministe markovienne.* Dans la figure (Figure 13 page 80), la meilleure (et seule) politique déterministe markovienne valide π_{det}, celle qui choisit l'action a_2, a la plus petite valeur pour ce MDP.
- *Pour un SPC MDP, il est possible qu'aucune politique optimale n'existe.* Pour la politique déterministe et histoire-dépendante $\pi^i_{d.h.}$ de la figure (Figure 13 page 80), où la politique $\pi^i_{d.h.}$ répète l'action a_1 i fois et choisit ensuite l'action a_2, plus i est grand et plus la politique a de récompense. De manière similaire, pour la politique markovienne aléatoire $\pi_{r.M.}$, la récompense accumulée par la politique augmente lorsque la probabilité $\pi_{r.M.}(A, a_2)$ tend vers 0.

Ce contre exemple prouve donc qu'il est trivial de construire un SPC MDP tel qu'aucune conversion en MDP ne permette de trouver une solution optimale qui puisse être convertie en solution optimale de ce SPC MDP, pour la raison simple qu'aucune solution optimale n'existe dans l'espace des politiques stochastiques et histoire-dépendantes.

Redéfinition de l'optimalité

L'une des solutions pour contourner ce problème est de définir néanmoins une notion d'optimalité et de rechercher la meilleure politique valide à un epsilon près. Formellement :

Définition 24 (*Epsilon-optimalité*)

Soit un processus de décision markovien à chemins contraints $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, s_0, \gamma, \xi \rangle$,
Soit $\Pi_{|\xi}$ *l'ensemble des politiques stochastiques valides,*
Soit $\epsilon \in \mathbb{R}^+ \setminus \{0\}$ *un réel strictement positif,*

Une politique π *est dite* solution ϵ-optimale d'un PCMDP *si elle vérifie :*

$$\pi \in \Pi_{|\xi}$$

$$\forall \pi' \in \Pi_{|\xi}, V_\gamma^\pi(s) \geq V_\gamma^{\pi'}(s) - \epsilon$$

Notre problème n'est donc plus de trouver la politique valide optimale, puisqu'elle n'existe pas toujours, mais de trouver une politique valide et epsilon-optimale, c'est-à-dire meilleure que les autres à epsilon près. Une telle politique existe toujours dès lors que la fonction de valeur V_γ^π est toujours finie. Ceci est en particulier le cas lorsque toutes les récompenses sont finies et que $\gamma < 1$, puisqu'on peut simplement borner V_γ^π.

L'exemple (Figure 13 page 80) nous donne ainsi deux intuitions sur la nature des solutions d'un SPC MDP, qui nous permettent de nous orienter vers la recherche de solutions avec une forme particulière :

- *1) La valeur de certaines politiques valides est arbitrairement proche de la valeur optimale.* Dans l'exemple ci dessus, les politiques qui effectuent une boucle sur l'état A pendant très longtemps puis se rendent en F ont des valeurs qui sont proches de $\frac{1}{1-\gamma}$, c'est-à-dire qu'elles sont "presque optimales" selon la définition (Définition 24 page 82).
- *2) Les meilleures politiques valides pour un SPC MDP ressemblent à certaines politiques déterministes Markoviennes.* Dans l'exemple ci-dessus, la meilleure (mais invalide) politique déterministe effectue une boucle sur l'état A pour un temps infiniment long, tandis que les politiques valides ϵ-optimales effectuent une boucle en A pour un temps "très" long avant d'effectuer une transition vers F.

Dans la suite de ce chapitre, nous allons prouver que ces intuitions sont correctes pour les modèles SPC MDP en général : pour tout $\epsilon > 0$, tout SPC MDP a une politique stochastique valide ϵ-optimal qui est similaire à une politique déterministe (mais invalide) pour ce MDP.

IV.2 Implémentation et preuve de validité d'un algorithme de résolution en programmation dynamique

Notre algorithme, permettant de trouver des politiques ϵ-optimales pour un SPC MDP, repose sur les intuitions précédentes. Il est constitué de deux étapes de base :

1. une opération de suppression, qui vise à compiler les contraintes du SPC MDP sous la forme de restrictions sur les états et chemins atteignables. Cette étape retourne un IHDR MDP pour lequel il existe des politiques ϵ-optimales qui sont valides et ϵ-optimales pour le SPC MDP original.
2. une procédure similaire à l'algorithme Value Iteration (Algorithme 3 page 76), qui permet de chercher une politique ϵ-optimale particulière associée à cet IHDR MPD.

La validité de l'algorithme complet repose sur plusieurs théorèmes, que nous détaillerons au cours des paragraphes suivants. Dans le contexte de cette preuve, il est plus logique de commencer par définir la seconde partie de l'algorithme, similaire à l'algorithme de Value Iteration. Le coeur de cet algorithme repose sur la recherche d'un type particulier de politique, que nous définissons de la manière suivante :

Définition 25 (*ω-politiques*)

On note $A(s)$ un ensemble d'actions possibles dans un état s d'un processus décisionnel markovien, avec $|A(s)| \geq 1$. Pour un réel $\omega \in [0,1]$ fixé, une ω-politique *est une politique markovienne aléatoire qui, pour chaque état s, choisit une des actions $a \in A(s)$ avec une probabilité $(1-\omega)$ (ou une probabilité 1 lorsque a est la seule action possible), et choisit chacune des autres actions de $A(s)$ avec une probabilité uniforme $\frac{\omega}{|A(s)|-1}$ (ou 0 si a est la seule action).*

Les ω-politiques correspondent bien à l'intuition #2 évoquée précédemment : lorsque ω est petit, une ω-politique se comporte de manière très similaire à une politique déterministe. En faisant tendre ω vers 0, on peut donc à partir de toute politique déterministe construire un ensemble de politiques aléatoires markoviennes qui convergent vers elle.

Il est important de noter qu'il y a plusieurs raisons pratiques de choisir des politiques aléatoires markoviennes plutôt que de choisir des politiques histoire-dépendantes avec un horizon fini (mais très large) : chercher une solution avec un horizon fini arbitrairement grand revient à résoudre un MDP à horizon fini ayant un horizon immense ; tous les algorithmes connus permettant de résoudre de tels processus décisionnels markoviens ont une complexité en temps de calcul qui est exponentielle en fonction de l'horizon. Ce facteur est limitant aussi bien pour des raisons pratiques que pour des raisons théoriques. Cet argument est utilisé pour justifier le choix d'un horizon infini dans d'autres classes de MDP où des états doivent être visités lorsque $t \to \infty$, par exemple les SSP [Ber95] et GSSP [KMWG11].

IV.2.1 Adaptation de l'opérateur de Bellman aux omega-politiques

Nous avons défini au paragraphe précédent une classe de politiques qui a des propriétés intéressantes vis-à-vis du problème SPC MDP. Parmi ces propriétés, nous pouvons montrer que pour un ω fixé, l'ω-politique optimale peut-être trouvée facilement :

Définition 26 (*Opérateur de Bellman pour les ω-politiques*)

Soit V une fonction de valeur pour un IHDR MDP, s un état et $A(s)$ un ensemble d'actions possibles dans l'état s, avec $|A(s)| > 1$. On définit l'opérateur ω-Bellman par $\omega Bell_{A(s)}V(s) =$

$$\max_{a \in A(s)} \left[(1-\omega) \left(\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{T}(s,a,s') V(s') \right) + \sum_{a' \in A(s) \setminus \{a\}} \frac{\omega \left(\mathcal{R}(s,a') + \gamma \sum_{s'} \mathcal{T}(s,a',s') V(s') \right)}{|A(s)| - 1} \right]$$

L'opérateur ω-Bellman se réduit à l'opérateur de Bellman classique pour les états s où $A(s)$ a une seule action.

Cette définition est facilement compréhensible lorsqu'on la compare à la définition de l'opérateur de Bellman classique :

Définition 27 (*Opérateur de Bellman (rappel)*)

Soit V une fonction de valeur pour un IHDR MDP, s un état et $A(s)$ un ensemble d'actions possibles dans l'état s. L'opérateur de Bellman est défini par :

$$Bell_{A(s)}V(s) = \max_{a \in A(s)} \left(\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{T}(s,a,s') V(s') \right)$$

De cette définition, on peut voir que l'opérateur ω-Bellman est simplement l'opérateur de Bellman appliqué à la forme particulière des ω-politiques, en prenant en compte la définition de la fonction de valeur pour les politiques stochastiques (Définition 19 page 25). Plus précisément, on peut montrer le résultat suivant pour cet opérateur :

Théorème 4 (*Convergence de l'opérateur ω-Bellman*)

Pour tout $\omega \in [0;1]$, l'opérateur ω-Bellman appliqué un nombre infini de fois à chaque état de l'IHDR MDP a un point fixe unique. Ce point fixe V^ω a la valeur la plus élevée parmi toutes les ω-politiques pour cette valeur de ω.

Démonstration : (1) Soit M un IHDR MDP. On peut construire un MDP M' en modifiant l'espace d'actions de M de la manière suivante : soit $A(s)$ l'ensemble de toutes les actions possibles dans un état s de M et $A'(s)$ l'ensemble des actions possibles dans un état s de M'.

Pour tout état s et action $a \in A(s)$ de M, on crée une action a' dans $A'(s)$ pour M' telle que la fonction de transition $T'(s,a',s')$ de M' est donnée en choisissant l'action a avec une probabilité $(1-\omega)$ et en choisissant toutes les autres actions avec une probabilité uniforme $\frac{\omega}{|A(s)|-1}$:

$$\forall s' \in S, T'(s,a',s') = (1-\omega)T(s,a,s') + \frac{\omega}{|A(s)|-1} \sum_{a_2 \in A(s)} (T(s,a_2),s') \tag{IV.2}$$

Cette construction crée une bijection entre toute action $a \in A(s)$ et l'action correspondante $a' \in A'(s)$. On construit la fonction de récompense R' pour M' de manière identique.

M' est aussi un IHDR MDP, tout comme M, et par construction appliquer l'opérateur classique de Bellman sur M' est équivalent à appliquer l'opérateur ω-Bellman sur M. Puisque l'opérateur classique de Bellman a un point fixe unique pour tout IHDR MDP, la fonction de valeur optimale [Put94], c'est par conséquent aussi le cas de l'opérateur ω-Bellman pour M.

(2) Toute ω-politique π pour M est associée à une politique déterministe π' pour M' au travers de la bijection entre les espaces d'actions. Il est trivial de montrer que π et π' ont la même valeur dans chaque état : il suffit d'injecter la fonction de valeur V de π dans la définition de la fonction de valeur de M' pour π' (Définition 19 page 25) ; il est immédiat de voir que V est un point fixe pour cette définition, ce qui montre que V est aussi la fonction de valeur de π'. Ceci définit donc une bijection entre les ω-politique pour M est les politiques déterministes pour M'.

La politique réalisant le point fixe π^* de l'opérateur ω-Bellman a alors la plus grande valeur entre toutes les ω-politiques, pour ω fixé, puisque la construction garantit qu'aucune autre ω-politique n'a de valeur supérieure en aucun état : on peut montrer facilement que, puisque la fonction de valeur de π^* est point fixe de l'opérateur de Bellman pour M', alors l'image de π^* par la bijection entre politiques est la politique déterministe optimale pour M' π'^*. Donc, par l'absurde, si une ω-politique π a une fonction de valeur V strictement supérieure dans l'état initial au point fixe, ceci serait aussi le cas en transposant les politiques dans M' - elle aurait une valeur strictement supérieure au point fixe - ce qui est impossible. Cette démonstration par l'absurde montre donc le résultat cherché. ■

Le théorème précédent permet donc de trouver la meilleure valeur d'une ω-politique pour ω fixé ; ceci permet également de trouver une ω-politique optimale en construisant une politique *gloutonne* associée, choisissant dans chaque état de mettre le poids $(1-\omega)$ à la meilleure action :

Définition 28 (*ω-politique gloutonne*)

Soit V^ω le point fixe de l'opérateur ω-Bellman pour une valeur ω fixée et un IDHR MPD donné. On dit que π^ω est une ω-politique gloutonne si pour chaque état s il existe une action a qui maximise la partie droite de l'opérateur ω-Bellman en s, et $\pi^\omega(s,a) = 1-\omega$.

Enfin, le théorème suivant justifie notre choix de nous concentrer sur la recherche d'ω-politiques pour trouver des solutions ϵ-optimales :

Théorème 5 (*Détermination du paramètre ω avec garanties d'ϵ-optimalité*)

Pour un IHDR MDP, $\gamma < 1$ et $\epsilon > 0$ donnés, on fixe : $\omega = \frac{\epsilon(1-\gamma)^2}{R_{max}-R_{min}}$ où R_{max} et R_{min} sont respectivement le maximum et le minimum de la fonction de récompense sur toutes les paires (état,action).

Cette valeur de ω garantit que toute ω-politique gloutonne est ϵ-optimale.

Démonstration : Pour un IHDR MDP M, Soit V^ω le point fixe de l'opérateur ω-Bellman, et soit V^* le point fixe de l'opérateur de Bellman classique sur ce même IHDR MDP.

Soit π^* la politique déterministe markovienne gloutonne vis-à-vis de V^* [Put94], et soit π^ω l'ω-politique gloutonne vis-à-vis de V^ω.

On utilise comme ω-politique intermédiaire la politique $\hat{\pi}$ qui dans chaque état associe le poids $(1-\omega)$ sur l'action $\pi^*(s)$.

On note $\hat{V}$ sa fonction de valeur. Puisque V^ω est la valeur maximale possible en tout point entre toutes les valeurs des ω-politiques, on a : $\forall s, \hat{V}(s) \leq V^\omega(s)$.

De manière similaire, puisque V^* est la valeur la plus haute possible entre toutes les politiques, y compris parmi les ω-policies, on peut écrire :

$$\forall s, \hat{V}(s) \leq V^\omega(s) \leq V^*(s) \tag{IV.3}$$

Pour borner la perte provoquée par ω, on note que pour toute fonction de valeur V et tout état s, on a l'inégalité suivante : $\frac{R_{min}}{1-\gamma} \leq V(s) \leq \frac{R_{max}}{1-\gamma}$. Ce résultat découle directement de la définition de la fonction de valeur (Définition 19 page 25).

On peut à présent utiliser l'équation de l'opérateur de Bellman classique (Définition 27 page 84) pour écrire, pour tout s :

$$\begin{aligned}
&|V^*(s) - \hat{V}(s)| \\
&\quad = \mathcal{R}(s,\pi^*(s)) + \gamma \sum \mathcal{T}(s,\pi^*(s),s')V^*(s') \\
&\quad - (1-\omega)\left[\mathcal{R}(s,\pi^*(s)) + \gamma \sum \mathcal{T}(s,\pi^*(s),s')\hat{V}(s')\right] \\
&\quad - \sum_{a \in A(s)\setminus\pi^*(s)} \frac{\omega}{|A|-1}\left[\mathcal{R}(s,a) + \gamma \sum \mathcal{T}(s,a,s')\hat{V}(s')\right] \\
&\quad \leq \omega R_{max} + \gamma \left|V^* - \hat{V}\right|_\infty + \omega\gamma\frac{R_{max}}{1-\gamma} - \omega R_{min} - \omega\gamma\frac{R_{min}}{1-\gamma}
\end{aligned}$$

La valeur de ω que nous cherchons découle donc en écrivant le résultat ci-dessus de la manière suivante :

$$|V^* - \hat{V}|_\infty \leq \omega\frac{R_{max}-R_{min}}{(1-\gamma)^2} \tag{IV.4}$$

On utilise finalement l'équation IV.3 : $|V^* - V^\omega|_\infty \leq |V^* - \hat{V}|_\infty$ ■

En d'autre termes, pour un paramètre ϵ fixé, ce théorème (Théorème 5 page 85) prouve l'existence d'une ω-politique ϵ-optimale. Le théorème (Théorème 4 page 84) nous permet alors d'en trouver une. Ces résultats sont à la base de la procédure d'itération de la valeur décrite dans l'algorithme de résolution (Algorithme 4 page 87).

Il est cependant important de noter que les théorèmes ci-dessus nous permettent de trouver une solution ϵ-optimale pour tout problème IHDR MDP, et non pour un SPC MDP. Il nous faut à présent décrire une étape permettant de convertir tout SPC MDP en une instance de IHDR MDP pour laquelle toute solution ϵ-optimale est valide et ϵ-optimale pour le problème d'origine.

IV.2.2 Pré-traitement spécifique aux contraintes non-transitoires ou à horizons finis

Comme nous l'avons vu, l'exemple de la figure (Figure 13 page 80) suggère que la valeur d'une politique peut être améliorée dès lors qu'on prend en compte l'historique de l'exécution du système dans un SPC MDP. Cependant, pour certains SPC MDP, on peut montrer facilement que la dépendance à l'histoire du système n'est pas nécessaire, et qu'il existe au moins une politique ϵ-optimale markovienne pour tout $\epsilon > 0$. C'est en particulier le cas lorsque toutes les fonctions booléennes qui sont utilisées dans les contraintes de ce SPC MDP sont *transitoires* [TK12a] (Figure 15 page 86) :

Définition 29 (*Fonction booléenne transitoire*)

Une fonction booléenne $f : \mathcal{S} \to \{true, false\}$ sur les états d'un MDP est transitoire *s'il existe des transitions depuis l'ensemble des états où $f(s) = true$ vers l'ensemble des états où $f(s') = false$, mais qu'il n'existe aucune transition dans la direction inverse, ou vice-versa.*

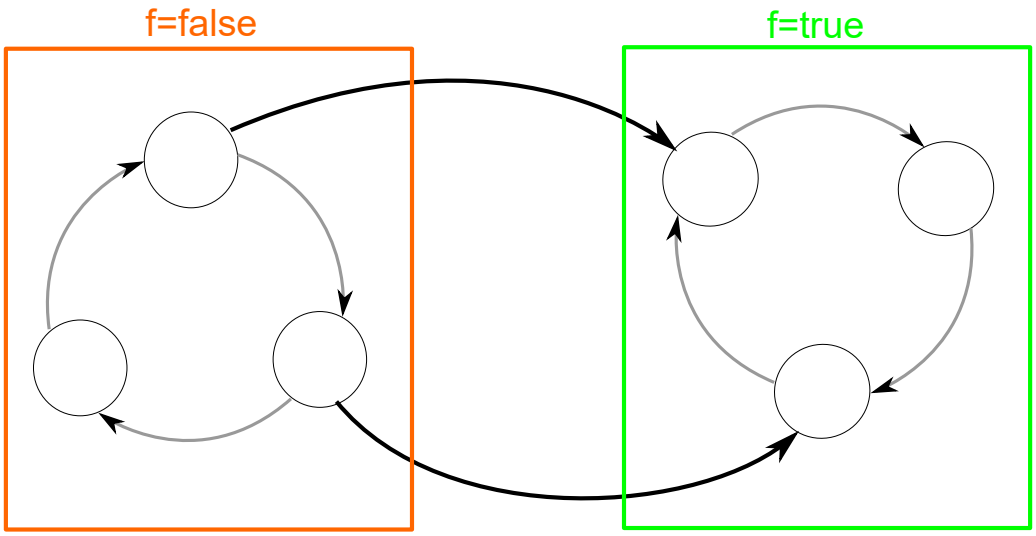


FIGURE 15 – Illustration des fonctions transitoires : f partage les états en deux ensembles disjoints, ayant des transitions dans un seul sens

Il est néanmoins possible de transformer des problèmes où certaines fonctions de certaines contraintes ne sont pas transitoires en des SPC MDP comprenant seulement des fonctions transitoires.

Théorème 6 (*Réduction aux fonctions transitoires*)

Soit $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, s_0, \gamma, \xi \rangle$ un SPC MDP, pour lequel m parmi n contraintes $\xi = \{\xi_1, \ldots, \xi_n\}$ comprennent des fonctions booléennes non transitoires. M peut être transformé en un SPC MDP $M' = \langle \mathcal{S}', \mathcal{A}, \mathcal{R}', \mathcal{T}', s'_0, \gamma, \xi' \rangle$ comprenant uniquement des fonctions transitoires en augmentant l'espace d'états avec une variable par contrainte non transitoire. Cette variable a trois valeurs possibles : (validée, invalidée, indéterminée). Le SPC MDP M' obtenu a ainsi $3^m|\mathcal{S}|$ états.

Démonstration : On étend l'espace d'états avec de nouvelles variables z_i : pour la contrainte i, cette variable devient 1 (i.e. "validée") lorsque le système entre pour la première fois dans un état où g devient vraie ; elle devient -1 (i.e. "invalidée") lorsque le système entre pour la première fois dans un état où f devient fausse. Par défaut, dans l'état initial et en l'absence de changements, elle vaut 0 (i.e. "indéterminée").

On définit la fonction "δ" comme un vérificateur sémantique pour cette extension : $\delta_s^{(i)}((s,z),(s',z')) =$

$$\begin{cases} true & if \quad (z_i = 0)\ \&\ g_i(s')\ \&\ (z'_i = 1) \\ true & if \quad (z_i = 0)\ \&\ !g_i(s')\ \&\ !f_i(s')\ \&\ (z'_i = -1) \\ true & if \quad \text{none above and } (z_i = z'_i) \\ false & else \end{cases}$$

On modifie la fonction de transition de manière correspondante : on définit $\mathcal{S}' = \mathcal{S} \times \{-1, 0, 1\}^m$ le nouvel espace d'états et $T'((s,z),a,(s',z')) = T(s,a,s')$ si et seulement si tous les $\delta_s^{(i)}((s,z),(s',z'))$ sont vrais. $\mathcal{R}'$,$\mathcal{I}'$ et ξ' sont les fonctions étendues naturellement à $\mathcal{S}'$, en omettant la partie en z de l'état.

Alors $\langle \mathcal{S}', \mathcal{A}, \mathcal{R}', \mathcal{T}', \mathcal{I}', \gamma, \xi' \rangle$ est un SPC MDP à contraintes transitoires : puisque la somme des issues possibles de la fonction de transition T reste égale à 1, on garde effectivement un SPC MDP correct. Notons qu'en raison de l'augmentation de l'espace d'états, toute politique histoire-indépendante optimale pour M' doit être transformée en politique histoire-dépendante, qui est optimale pour M. ■

Ce résultat constitue l'étape préliminaire de notre algorithme permettant de résoudre un SPC MDP, que nous détaillons dans la section suivante. Cette étape permet d'éliminer les fonctions non-transitoires, puis évalue les contraintes dans un passage préliminaire en supprimant les états et actions qui ne les respectent pas de façon évidente.

Notons que bien que cette étape augmente l'espace d'états de façon linéaire en fonction des données d'entrées, il existe des cas où elle est potentiellement une des étapes les plus coûteuses en termes de complexité : comme nous le verrons dans les paragraphes suivants, notre algorithme a une complexité polynomiale en fonction des données d'entrées lorsque l'espace d'états est énuméré. Cependant, ce résultat de complexité n'est plus valide dès lors que l'on considère d'autres formats de données d'entrées, tels que les données d'entrées du MDP d'origine. Ceci est expliqué par l'une des deux raisons suivantes :

1. soit par le fait que la conversion d'un problème quelconque en une instance de SPC MDP nécessite un temps de traduction non-polynomial ou un espace d'états polynomial en fonction de la taille des données,
2. soit par l'étape préliminaire de traduction : pour appliquer notre algorithme, il est nécessaire que toutes les fonctions booléennes soient transitoires, et l'étape de traduction peut nécessiter d'augmenter l'espace d'états de façon polynomiale, en particulier lorsque le nombre de contraintes PCTL à créer dépend directement du nombre d'états.

Cependant, lorsque le nombre de contraintes PCTL est négligeable par rapport au nombre d'états, cette étape de traduction a un impact mineur sur le temps de résolution.

IV.2.3 Description et preuve de validité de l'algorithme

L'algorithme que nous construisons sur ce schéma est appelé SPC VI pour Saturated Path Constraints Value Iteration. La boucle principale est présentée dans l'algorithme suivant (Algorithme 4 page 87).

Algorithm 4: SPC Value Iteration

Procédure SPC Value Iteration $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, s_0, \gamma, \xi, \epsilon, \theta)$;
Entrées : $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, s_0, \gamma, \xi \rangle$ tels que définissants un SPC MDP avec contraintes $\xi_1, \ldots, \xi_n$
$\epsilon > 0$ un paramètre d'optimalité
$\theta > 0$ un paramètre de convergence
Sorties : π ω-politique ϵ-optimale
Variables : $\hat{\mathcal{S}}$ l'ensemble des états explorés
Tip un sous-ensemble des états explorés non étendu

Convertir les fonctions non-transitoires des contraintes (Théorème 6 page 86) si nécessaire ;
Initialiser $\hat{\mathcal{S}} := \{s_0\}$;
Initialiser $Tip := \{s_0\}$;
$explore(\hat{\mathcal{S}}, Tip)$;
for $i : 1 \ldots n$ **do** $updateReachability(\hat{\mathcal{S}}, \xi_i, \theta)$;
for $s \in \mathcal{S}$ **do** $A(s) := \left\{ a \mid \forall s', T(s, a, s') > 0 \Rightarrow s' \in \hat{\mathcal{S}} \right\}$;
Initialiser $\omega := \frac{\epsilon(1-\gamma)^2}{R_{max} - R_{min}}$;
Calculer V^ω par Value Iteration avec l'opérateur $\omega Bell$;
Retourner π ω-politique gloutonne vis-à-vis de V^ω

Algorithm 5: Fonction $explore(\hat{\mathcal{S}}, Tip)$

Fonction *explore* $(\hat{\mathcal{S}}, Tip)$;
Entrées : $\hat{\mathcal{S}}$ un espace des états explorés
Tip un espace des états explorés et non étendus
Sorties : $\hat{\mathcal{S}}$ espace des états explorables par toutes les politiques non trivialement invalides

repeat
  Choisir $s \in Tip$;
  if *il y a une contrainte* $\xi_i = f_i U^{\infty}_{=p_i} g_i$ *telle que* $((p_i = 1 \,\&\, !f_i(s))$ *ou* $(p_i = 0 \,\&\, g_i(s)))$ **then**
    Retirer s de Tip et $\hat{\mathcal{S}}$;
  else
    for *a et* s' *tels que* $T(s, a, s') > 0$ **do**
      if s' *n'est pas déjà dans* $\hat{\mathcal{S}}$ **then**
        Ajouter s' à Tip et $\hat{\mathcal{S}}$
until $Tip = \emptyset$;

Algorithm 6: Fonction $updateReachability(\hat{\mathcal{S}}, \xi_i, \theta)$

Fonction *updateReachability*$(\hat{\mathcal{S}}, \xi_i, \theta)$;
Entrées : $\hat{\mathcal{S}}$ un espace d'états
ξ_i une contrainte de la forme $\xi_i = f_i U^{\infty}_{=p_i} g_i$
θ un paramètre de convergence
Sorties : $\hat{\mathcal{S}}$ espace des états valides
Variables : X et X' des fonctions de valeur représentant la validité

Initialiser X, X' à 0;
for $s \in \hat{\mathcal{S}}$ **do** $X(s) := g_i(s)$;
repeat
  $X' := X$;
  for $s \in \hat{\mathcal{S}}$ **do**
    if $p_i = 1$ **then**
      if $!f_i(s)$ **then**
        $X(s) := 0$
      else
        $X(s) := \max_a \left[\sum_{s'} \mathcal{T}(s, a, s') X(s') \right]$
    if $p_i = 0$ **then**
      if $g_i(s)$ **then**
        $X(s) := 1$
      else
        $X(s) := \min_a \left[\sum_{s'} \mathcal{T}(s, a, s') X(s') \right]$
until $\mid X - X' \mid_{max} < \theta$;
for $s \in \hat{\mathcal{S}}$ **do**
  if $((p_i = 1) \,\&\, X(s) < 1 - \theta)$ *ou* $((p_i = 0) \,\&\, X(s) > \theta)$ **then** Retirer s de $\hat{\mathcal{S}}$ et supprimer les actions y menant ;

Cet algorithme est constitué de deux étapes :

Étape 1 : Traitement des contraintes

Dans la première étape, l'algorithme effectue un pré-traitement sur les fonctions non-transitoires des contraintes (Théorème 6 page 86). Il identifie ensuite les états du SPC MDP qui ne peuvent pas être atteints depuis l'état initial sans violer au moins une des contraintes (Algorithme 5 page 88) (l. 6), les états dans lesquels au moins une contrainte n'est pas satisfaisable, et toutes les actions menant à ces types d'états (Algorithme 6 page 88) (l. 12 et 17). Comme le montre l'algorithme (Algorithme 6 page 88), on peut effectuer cette recherche de manière itérative par une technique simple de programmation dynamique. Ces actions et états ne peuvent pas faire partie d'une politique valide, ils sont donc retirés du MDP (l. 20), ce qui laisse un espace d'états $\hat{\mathcal{S}}$ réduit et un espace d'actions $A(s)$ pour chaque $s \in \hat{\mathcal{S}}$.

On peut résumer cette partie de l'algorithme de la manière suivante :

- $\hat{\mathcal{S}}$ est obtenu après la fonction *explore* function, où les états qui violent de manière évidente au moins une contrainte sont mis de côtés.
- A est l'ensemble des actions "valides" et est défini après la fonction *updateReachability*, dans laquelle une opération de Programmation Dynamique est appliquée successivement pour chacune des contraintes, afin de calculer la valeur (vis-à-vis de la validité) pour la meilleure politique possible dans chaque état. Cette opération est inspirée par [HJ94], où elle est utilisée pour calculer la valeur d'une politique donnée. Les preuves de terminaison et de validité demeurent les mêmes.

Ces deux espaces d'états et d'actions forment un IHDR MDP avec les propriétés suivantes :

Lemme 5

Pour un SPC MDP, soit Π_A l'ensemble de toutes les politiques possibles sur l'ensemble des états $\hat{\mathcal{S}}$ tels que définis ci-dessus, tel que toute politique $\pi \in \Pi_A$ assigne une probabilité 0 à toute action $a \notin A(s)$ pour tout état $s \in \hat{\mathcal{S}}$ que cette politique atteint.

1. *Π_A contient toutes les politiques valides.*
2. *Toute ω-politique avec $\omega > 0$ dans Π_A est valide.*

Démonstration : A décrit un ensemble très large de politiques : une action est dans A si et seulement si il existe un chemin démarrant par cette action qui respecte toutes les contraintes. Ce chemin peut être histoire-dépendant, ce qui implique qu'on décrit un ensemble de politiques plus large que l'ensemble des solutions.

La preuve du résultat **(1)** résulte du fait que toute politique à l'extérieur de Π_A doit utiliser une action $a \notin A(s)$ dans au moins un état s ; mais une telle action est invalide, ce qui signifie qu'il n'y a pas de politique aléatoire histoire-dépendante qui valide toutes les contraintes en choisissant l'action a. Par conséquent, une telle politique ne peut pas non plus être valide.

Le résultat **(2)** découle du fait que puisque $\hat{\mathcal{S}}$ ne contient aucun état ou action invalide, toute politique qui effectue une marche aléatoire sur $\hat{\mathcal{S}}$ finira par satisfaire toutes les contraintes de ce SPC MDP.

En donnant plus de détails : si on avait une ω-politique invalide dans cet ensemble, ce serait une politique choisissant uniquement des actions dans $A(s)$ et choisissant au moins chaque action avec une probabilité ω. Puisqu'elle est invalide, on peut affirmer l'une des deux propositions suivantes :

- cette politique atteint un état non autorisé ("f est vraie et p=1" ou "g est vraie et p=0") pour au moins une contrainte, ce qui est impossible puisque l'action correspondante qui nous a fait atteindre cet état ne serait pas dans $A(s)$.
- cette politique violerait une contrainte "$p = 1$" et n'atteindrait jamais un état où g devient vraie ; ce qui est impossible puisque pour tout état s accessible à partir de l'état initial, il existe un chemin de n état et n actions de A qui mènent à un état s' où $f(s')$ est vraie (cette existence est garantie avec une probabilité strictement supérieure à 0, sinon la récompense 1 n'aurait pas été propagée vers cet état s), et que la probabilité de choisir ce chemin est au moins de ω^n. ■

Par conséquent, la sortie de la première étape de l'algorithme est un nouvel espace d'états et un nouvel espace d'actions à partir desquels sont construits toutes les politiques valides possibles, y compris les ω-politiques. Ceci signifie que *s'il existe une ω-politique ϵ-optimale valide*, celle-ci est nécessairement construite à partir de ces "briques".

Étape 2 : Value Iteration basée sur l'opérateur ω-Bellman

La seconde étape de l'algorithme (Algorithme 4 page 87) (l. 8-10) commence par la détermination d'une valeur de ω (l. 9) pour lequel toutes les ω-politiques gloutonnes sur l'espace d'états et l'espace d'actions restants sont ϵ-optimales, via le théorème prouvé précédemment (Théorème 5 page 85). L'algorithme itère alors *l'opérateur ω-Bellman* (l. 10) pour trouver une ω-politique gloutonne qui est ϵ-optimale pour le MDP avec les espaces d'états et d'actions réduits. Par construction, comme ce MDP contient toutes les politiques valides, cette politique est ϵ-optimale parmi toutes les politiques valides pour le MDP complet. Avec le lemme 5, cette politique est également valide, ce qui nous permet d'obtenir le résultat suivant :

Théorème 7 (*Existence d'ω-politiques ϵ-optimales*)

Pour tout SPC MDP avec des contraintes satisfiables, il existe une politique ϵ-optimale valide et l'algorithme SPC VI permet de la trouver. La solution est aléatoire et markovienne dans l'espace d'états augmenté, mais histoire-dépendante dans l'espace d'états d'origine.

Ce résultat fondamental montre ainsi la validité de notre algorithme. Dans la section suivante, nous allons à présent évaluer les performances de cet algorithme sur un ensemble de cas de tests académiques, en particulier en comparant le temps de calcul et la qualité de la solution vis-à-vis des algorithmes existants pour PCMDP.

IV.3 Évaluation des performances de l'algorithme sur un ensemble de cas de tests académiques

Nous fixons deux objectifs à cette évaluation :

1. Comparer SPC VI avec l'algorithme PCMDP-ILP [TK12a] en termes d'efficacité. Cette comparaison permettra de savoir dans quelle mesure le passage de PCMDP à SPC MDP permet effectivement de diminuer la complexité du problème, le rendant possible pour des problèmes de taille plus importante.
2. Valider SPC MDP en tant que modèle pour représenter des problèmes. Nous souhaitons évaluer si les hypothèses supplémentaires que nous avons posées pour construire le cadre SPC MDP permettent néanmoins d'exprimer un panel suffisant de problèmes.

Ces expériences ont été menées sur un ordinateur ayant 5.8Gb de RAM et un CPU à 2.80GHz. Nous avons utilisé deux benchmarks [1] exprimées dans le langage PPDDL, étendu par l'ajout de contraintes PCTL. Dans toutes les expériences, le facteur de dévaluation γ a été fixé arbitrairement à $\gamma = 0.9$. Les tests que nous avons effectués ont montré que le choix d'un paramètre γ différent n'impactait le temps de résolution que de manière mineure sur les domaines considérés.

IV.3.1 Drone autonome de lutte contre les incendies

Comme premier exemple de scénario qui peut être traité avec le modèle SPC MDP, on considère la conception de plans de vol pour des drones automatiques de gestion de feux de forêts. Les drones doivent planifier leur mission de façon à respecter un ensemble de contraintes faibles et fortes. Parmi ces contraintes, on comptera notamment les suivantes :

- Le drone doit surveiller régulièrement des zones avec un taux élevé de risque de feux de forêts.
- Il doit remplir des objectifs isolés tels que la reconnaissance de zones d'anciens incendies.
- Il doit dans la mesure du possible occasionnellement effectuer des vérifications sur des zones d'importances secondaires.
- Il ne doit jamais survoler certaines régions non autorisées, telles que des zones résidentielles.

Ce scénario est intéressant en ce que des solutions satisfaisantes ne peuvent pas être obtenue par des outils plus simples que le modèle SPC MDP, tels que les modèles pouvant se ramener à un modèle MDP : bien que les contraintes faibles peuvent être représentées en MDP en associant des récompenses fortes à certains états, les contraintes dures sont plus délicates à modéliser : aucun MDP existant ne permet à la fois de traiter des buts non terminaux, des états cul-de-sac dans lesquels les contraintes ne peuvent pas être respectées, et des récompenses arbitrairement positives et négatives. Même les classes de MDP les plus générales, telles que les SSP MDP [Ber95] restreignent la structure de récompense, par exemple en forçant les récompenses à être négatives, et nécessitent que les buts soient absorbants.

Hypothèses de modélisation

Pour traiter le domaine du Drone avec un modèle SPC MDP, nous l'avons formulé de la manière suivante : Le drone évolue sur une grille, en se déplaçant selon quatre directions. La figure (Figure 16 page 94) montre un exemple de plan de vol. Il y a quatre types de zones possibles :

- Des zones à haut risque (rouge), où des incendies apparaissent de manière fréquente et qui doivent être surveillées au moins toutes les x heures.
- Des zones à risque plus faible (orange), où un incendie peut potentiellement démarrer. Dans la mesure du possible, le drone devrait surveiller ces zones régulièrement.
- Des objectifs obligatoires (bleu) qui doivent être survolés au moins une fois durant le vol, tels que les sites d'incendies déjà éteints que l'on souhaite contrôler.
- Des zones non-autorisées (noir) qui ne doivent jamais être survolées.

1. procédure de tests automatisés

Nous avons obtenu les probabilité d'occurrence d'incendies à partir de plusieurs sources statistiques [PBBB04]. Nous avons posé comme hypothèse qu'une fois qu'un drone a surveillé une zone à risque, la probabilité d'incendie descend à 0 pour les prochaines y heures. En effet, si le drone détecte un danger immédiat, ce danger peut être traité avant que le feu ait complètement démarré. Lorsqu'un incendie survient, nous supposons qu'il est éteint rapidement et qu'un autre incendie ne peut pas survenir immédiatement au même endroit.

On modélise les contraintes de zone à risque faible (type 2) avec des récompenses, comme pour un modèle MDP classique. On augmente l'espace d'états de base avec une variable temporelle, augmentant à chaque action (de la durée de l'action) et remise à zéro après chaque action de surveillance. Pour les zones à haut risque, nous avons deux possibilités de modélisation :

1. Modéliser un incendie dans une zone à haut risque par un cul de sac : lorsqu'un incendie survient suite à une transition aléatoire, plus aucune action ou transition n'est possible. Ainsi, aucun objectif obligatoire ne pourra être rempli, ce qui implique qu'un tel cul de sac viole les contraintes du SPC MDP.
2. Modéliser un tel incendie par une contrainte d'interdiction de la forme $trueU^{\infty}_{=0}g$ où g est vraie lorsqu'un incendie s'est produit.

Notons que la première solution n'est possible que parce qu'il existe au moins une contrainte obligatoire d'aliénabilité ; sans une telle contrainte, il faudrait pouvoir garantir que toutes les politiques aboutissant à ce cul de sac ont une valeur strictement plus faible que la meilleure politique possible n'atteignant pas un de ces culs-de-sac. Cette garantie est notablement difficile à obtenir, cette difficulté étant l'un des apports principal de cadre tels que GSSP [KMW12], permettant de traiter des problèmes SSP avec cul de sac.

Notons que pour la solution (2), il est préférable de faire une seule contrainte pour toutes les zones d'incendie, plutôt qu'une contrainte par zone : la première partie de notre algorithme est en effet de complexité linéaire en fonction du nombre de contraintes ; ce choix n'a en revanche pas d'impact sur la seconde partie de notre algorithme.

Enfin, le choix de la méthode de modélisation peut être laissé à la préférence du modélisateur : en termes de temps de calcul, la première option a un léger avantage dans le cas général en ce qu'elle nécessite une contrainte de moins ; cependant, l'algorithme a une complexité linéaire en fonction du nombre de contraintes, impliquant que le gain de temps est minime. Par ailleurs, la méthode 2 offre l'avantage de pouvoir définir les zones indépendamment de la carte (i.e. de la grille et des déplacements possibles), ce qui permet facilement de tester des contraintes sur des zones différentes, mais en gardant la même carte, par exemple en passant une "zone à risque modéré" en "zone à haut risque" sans modifier les transitions.

Cette séparation entre modèle et contrainte n'est cependant que idéologique : il y a bien évidemment de multiples manières de modéliser la définition des zones pour faciliter la modification du modèle, par exemple au travers de variables intermédiaires ; mais cette séparation est souvent bénéfique puisqu'elle permet de séparer le processus de conception du modèle en deux temps : la conception de la carte d'une part, par exemple à partir d'une base de données de forêts, puis la conception des zones à risque d'autre part, par exemple à partir de rapports écrits ou de connaissances métiers des autorités locales.

Pour les contraintes définissant des objectifs obligatoires, nous avons utilisé des contraintes de la forme $trueU^{\infty}_{=1}g$. Ces contraintes expriment que le drone doit nécessairement visiter durant son parcours un état où $g(s) = true$; la fonction g est alors vraie seulement pour les états de ces zones obligatoires. Notons que contrairement à d'autres cadres mathématiques tels que SSP [Ber95], le système ne s'arrête pas nécessairement après avoir rempli les objectifs demandés : il peut potentiellement continuer indéfiniment. Il est aussi possible de préciser un objectif cul de sac, comme par exemple un retour à la base ; l'algorithme de résolution devra nécessairement remplir cet objectif en dernier.

Pour les zones interdites, nous avons trois manières possibles de les modéliser :

1. Interdire le drone d'y entrer : toute action y menant échoue, et le drone reste sur place.
2. Mettre une pénalité forte, sous la forme d'une récompense négative, sur ces états.

3. Ajouter une contrainte d'interdiction, de la forme $trueU^{\infty}_{=0}g'$

Les deux premières approches sont des approches classiques de MDP ; elles ont des inconvénients qui ont déjà été étudiés largement dans la littérature, en ce qu'il n'est potentiellement pas facile de déterminer quelle est la conséquence de la modification d'une action, ou la conséquence de la modification d'une récompense : pour la première méthode, il faut s'assurer que la modification ou la suppression d'une action n'impacte pas la politique optimale. Pour la seconde méthode, il nous faut choisir une récompense suffisamment faible pour que la politique optimale ait une valeur strictement supérieure ; cependant, la valeur de la récompense à choisir est difficile à déterminer par avance, et est potentiellement indéterminée si aucune politique valide n'existe.

Pour ces raisons, nous avons choisi la troisième méthode dans notre modélisation : bien qu'elle soit marginalement moins efficace (puisque l'algorithme a une complexité linéaire en fonction du nombre de contraintes), elle est sensiblement plus simple à mettre en place et robuste aux changements de modèles.

Conditions de test

Nous avons généré de manière aléatoire des instances de ce problème pour des grilles de tailles différentes (de 10x10 à 100x100, avec un nombre total d'états entre 200 et 160 000), pour des paramètres de temps différents, et des positions, tailles, et nombre de zones de chaque type différent (entre 1 et 5 par type). Pour le paramètre d'ϵ-optimalité, nous avons fixé arbitrairement $\epsilon = 0.1$, avec un paramètre correspondant $\omega = 0.001$, puisque la pénalité de déclenchement d'incendie a été fixée à -1.

Nous avons testé d'autres ajouts, tels que la prise en compte d'un compteur pour modéliser une quantité de carburant restant, mais ces ajouts se sont avérés être trop complexes pour le solveur PCMDP-ILP avec lequel nous souhaitions nous comparer. Par ailleurs, de tels ajouts n'apportent que peu d'informations sur la validité du modèle SPC MDP ou ses performances ; ils permettent simplement de mettre en valeur l'expressivité du langage de modélisation choisi.

Comparaison avec l'algorithme PCMDP-ILP

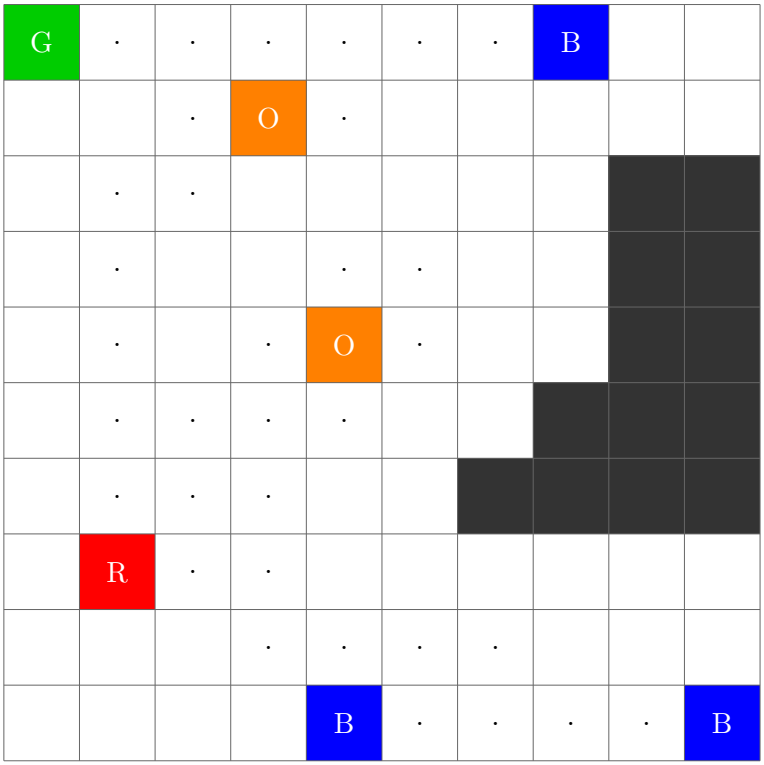
Nous avons utilisé ce domaine Drone pour évaluer les performances de SPC VI vis-à-vis de PCMDP-ILP [TK12a]. Cette comparaison est possible puisque ces deux algorithmes produisent le même type de solution (politiques ϵ-optimales, aléatoires et markoviennes lorsque toutes les fonctions sont transitoires).

L'une des motivations principales de notre travail a été de formuler un modèle MDP ayant des contraintes, mais étant plus efficace à résoudre que PCMDP, avec aussi peu de perte d'expressivité que possible. Comme le montre la figure (Figure 17 page 94), SPC MDP respecte bien ces critères. Le domaine peut en effet être exprimé dans chacun des deux types de MDP, mais SPC VI permet une résolution bien plus rapide que PCMDP-ILP sur toutes les instances du domaine que nous avons testées : sur la figure, chaque point correspond à une instance, et tous les points se trouvent en dessous de la ligne diagonale.

Plus précisément, l'algorithme SPC VI est plus rapide de *plusieurs ordres de grandeur* que l'algorithme PCMDP-ILP : si on dessine une droite reliant les points, cette droite a une croissance plus faible que la droite médiane ; puisque la figure est en échelle logarithmique, alors pour un problème tel que le domaine du drone où les temps de résolutions augmentent de façon exponentielle pour l'algorithme PCMDP-ILP (passant de 0.1 seconde à 1s puis 10s puis 100s puis 1000s...), les temps de résolution pour SPC VI augmentent avec un facteur exponentiel plus faible (passant de 0.1s jusqu'à un maximum de 1 pour les mêmes instances). Notons qu'en raison de la complexité du problème SPC MDP, la croissance du temps de calcul augmente néanmoins toujours de façon exponentielle avec la difficulté des instances.

IV.3.2 Gestion des mises à jour d'un parc de serveurs

Ce domaine décrit le déploiement d'une mise à jour importante sur tous les nœuds d'un cluster (i.e. un ensemble d'ordinateurs). Cette mise à jour est importante, mais n'est pas critique au point de



Vert : Départ
Orange : risque d'incendies
Rouge : haut risque, ne doit jamais être laissé sans surveillance
Bleu : cendres, doit être surveillé au moins une fois
Noir : non autorisé
Points : état visité par la politique optimale

FIGURE 16 – Exemple d'instance du domaine Drone

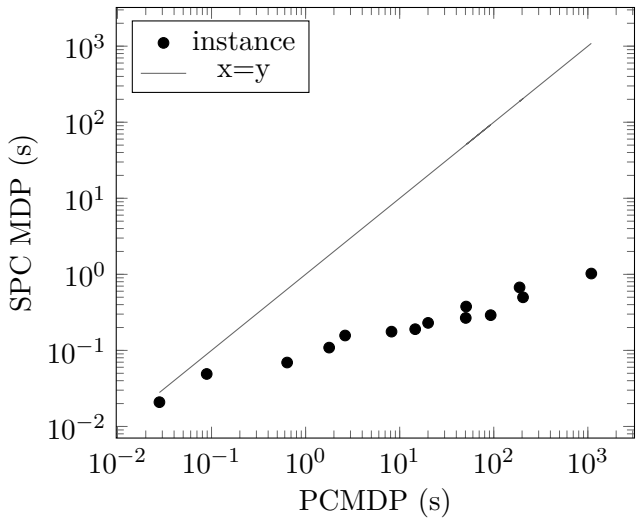


FIGURE 17 – Temps de résolution pour le domaine Drone (échelle logarithmique) : chaque point est une instance du problème.

nécessiter une mise à jour immédiate. La mise à jour peut être installée sur chaque nœud à une date différente, indépendamment des autres nœuds.

Chaque nœud a deux modes de fonctionnement, s_1 (basse énergie) et s_2 (haute énergie). À chaque heure passée dans le mode s_1, le nœud coûte r_1 unités ; nous pouvons par exemple fixer $r_1 = -10$ centimes. Chaque heure passée dans le mode à haute énergie s_2 coûte r_2 unités, par exemple -20 centimes. Chaque nœud peut individuellement changer de mode de fonctionnement une fois par heure, mais chacun de ces changements a un coût en énergie, par exemple de -30 centimes. Déployer la mise à jour prend 7 heures, durant lesquelles le nœud doit être dans le mode à haute énergie.

Ce domaine est rendu non trivial par la présence de l'utilisateur. À tout moment, chaque nœud est soit utilisé, soit inutilisé. À chaque heure où un nœud n'est pas utilisé, il y a une probabilité p qu'un utilisateur commence à l'utiliser. Si cela arrive, le nœud restera occupé pendant les 4 prochaines heures, au cours desquelles aucun autre utilisateur ne pourra utiliser ce nœud. Dans ce domaine, les jours sont divisés en périodes de 12 heures, correspondant au jour et à la nuit. La probabilité p qu'un nouvel utilisateur apparaisse dépend de la période : elle est de 1/4 en journée et de 1/8 pendant la nuit.

Les utilisateurs préfèrent que leur nœud soit dans un état de haute énergie ; ils payent 50 centimes pour chaque heure où ils utilisent un nœud dans un état à haute énergie (dont 20 centimes serviront à payer les coûts en énergie). Les utilisateurs n'apprécient pas lorsqu'un nœud est dans un état à basse énergie alors qu'ils l'utilisent ; ils ne payent que 10 centimes par heure dans ce cas. Ils apprécient encore moins lorsqu'un ordinateur subit une mise à jour alors qu'ils l'utilisent ; ceci coûte à l'administrateur -50 centime par heure (ajoutés au coût payé pour maintenir le système dans un état à haute énergie).

En résumé, pour chaque heure (étape de temps), il y a trois actions par nœud :

1. mettre à jour,
2. changer de mode de fonctionnement,
3. ne rien faire.

La contrainte est que chaque nœud doit être à un moment mis à jour et doit être dans le mode à haute énergie pour la durée de la mise à jour.

Conditions de test

Puisque la fonction de récompense varie de -50 à 50, on choisit un paramètre $\epsilon \leq 10$ pour le paramètre d'ϵ-optimalité. Le paramètre ω correspondant est 0.001. Nous avons testé des instances ayant de 1 (1246 états) à 3 ordinateurs (1 014 013 états).

ϵ	N	**t (s)**	**V (dévaluée)**
10	1	0.045	-19.6931
	2	5.95	-39.1865
	3	54.6	-60.1082
1	1	0.048	-19.1382
	2	5.82	-38.616
	3	53.8	-59.8614
0.1	1	0.041	-19.0826
	2	5.89	-38.5589
	3	54.5	-59.8367

TABLE 1 – Temps de résolution sur le domaine Mise à jour de serveurs

Influence du paramètre epsilon

Pour ce domaine, nous avons souhaité étudier l'influence du paramètre ϵ. Comme le montre le tableau (Tableau 1 page 95), il n'y a pas d'impact évident du choix d'une valeur ϵ plus petite sur le temps total de résolution. Ceci peut principalement être expliqué en notant que ϵ dans l'algorithme

SPC VI n'est pas utilisé comme paramètre de terminaison, comme il peut l'être dans l'algorithme VI pour les IHDR MDP, ou indirectement dans PCMDP-ILP. Il a donc une influence minime sur le temps de calcul de SPC VI.

Nous avons aussi cherché à évaluer les politiques obtenues sur le plan qualitatif. Le résultat que nous avons observé est que le système tendait à installer les mises à jour la nuit, juste après que le dernier "utilisateur du jour" ait fini d'utiliser un nœud en état de haute énergie. Il s'agit d'un résultat très intuitif : la nuit, le risque de dépenses engendrées par une interférence avec un utilisateur pendant une mise à jour est plus faible. De plus, en installant une mise à jour juste après le départ d'un utilisateur, alors que le nœud est toujours dans un état à haute énergie, le système évite de perdre des ressources en repassant dans un état à faible énergie, puis en revenant dans l'état à haute énergie plus tard pour satisfaire aux conditions nécessaires pour l'installation.

IV.3.3 Conclusion et résumé des apports sur le modèle SPC MDP

Dans les chapitres précédents, nous avions identifié plusieurs limitations des modèles existants [TK12a] permettant de traiter des problèmes de planification en environnement probabiliste sous contraintes de chemin, spécifiquement deux limitations : (1) certains modèles ne permettent pas de traiter des problèmes de taille industrielle et (2) les politiques solution obtenues par certains modèles ne permettent pas de garantir le niveau requis d'optimalité ou de respect des contraintes que nous espérons avoir.

Dans ce chapitre, nous avons étudié un nouveau modèle mathématique, permettant de pallier ces limitations, en réalisant plus particulièrement les apports suivants :

- Nous avons défini un nouveau modèle, **Saturated Path-constrained Markov Decision Process**, basé sur le modèle PCMDP avec la simplification de se limiter à un sous-ensemble de contraintes PCTL dites "saturées".
- Nous avons en particulier montré qu'il était nécessaire de définir une **ϵ-optimalité** pour cette classe de problème, nécessitant l'utilisation de politiques non-déterministes ou non-markoviennes pour s'approcher autant que possible d'une valeur optimale.
- Nous avons défini un **algorithme de résolution** pour les problèmes SPC MDP, en prouvant la validité et l'ϵ-optimalité de la solution obtenue ; ceci a en particulier nécessité la définition d'un opérateur inspiré de l'opérateur de Bellman.
- Nous avons enfin **évalué les performances de cet algorithme** sur plusieurs cas d'applications académiques, qui ont montré que sur un même modèle l'algorithme de résolution SPC VI a effectivement un temps de résolution plus faible de plusieurs ordres de grandeur par rapport aux algorithmes de résolutions pour PCMDP qui nous étaient accessibles.

Le modèle SPC MDP semble donc être un bon candidat pour la résolution du scénario d'aide à la décision pour la maintenance avionique, en ce qu'il permet de prendre en compte la dynamique probabiliste de l'environnement contrairement à des solutions de planification classique, tout en ayant une complexité raisonnable en termes de temps de calcul. Il est indéniable que le cadre SPC MDP apporte des garanties fortes sur la sécurité de la solution obtenue, avec les deux inconvénients suivants :

1. L'algorithme de résolution SPC MDP obtient une politique aléatoire, ce qui est contre-intuitif et pose le problème de l'acceptation et de la mise en œuvre de cet algorithme dans un contexte industriel.
2. La résolution effectue une exploration exhaustive de tous les futurs possibles, ce qui limite grandement la taille des modèles qu'il est possible de traiter.

Ces deux inconvénients dépendent fortement de l'application choisie : sur une application ne présentant aucune boucle, la politique solution pourra par exemple être réduite à une politique déterministe ; de la même manière, plus les contraintes sont difficiles à respecter plus l'algorithme pourra très rapidement arrêter l'exploration des futurs possibles, ce qui lui permettra de traiter des modèles de taille très importante.

Si le modèle SPC MDP remplit bien l'objectif que nous nous étions fixé - celui de combler un manque de méthode de résolutions efficaces pour PCMDP dans la littérature existante - il nous faut donc à présent évaluer dans quelle mesure ce modèle permet effectivement de traiter notre scénario d'aide à la décision pour la maintenance avionique.

CHAPITRE V

ÉVALUATION DE LA CAPACITÉ DU MODÈLE SPC MDP À RÉSOUDRE LE PROBLÈME DU BUSINESS JET

Sommaire

DANS le chapitre précédent, nous avons montré que notre algorithme SPC VI permettait de résoudre tout problème de décision sous contrainte de logique PCTL qui est exprimé sous la forme d'un modèle SPC MDP. Nous avions construit le modèle SPC MDP comme modèle d'expression de notre problème puisque nous souhaitions considérer un modèle avec une dynamique probabiliste, mais sans affronter le niveau de complexité engendré par des modèles plus complets tels que PCMDP [TK12a].

Notre objectif à présent est alors double :

1. Nous souhaitons évaluer si les hypothèses que nous avons choisies sur le modèle SPC MDP sont correctes, c'est-à-dire qu'elles ne sont pas trop restrictives. Pour cela, nous allons chercher à exprimer notre scénario de maintenance d'un avion de type business jet sous la forme d'un SPC MDP.
2. Nous souhaitons évaluer si un processus outillé basé sur SPC MDP est possible dans le cas d'application envisagé. Ceci implique de développer un prototype de ce processus outillé, c'est-à-dire de construire sur notre scénario restreint toutes les étapes depuis les documents et connaissances disponibles en entrée jusqu'aux résultats attendus.

Au cours de ce chapitre, nous allons définir les étapes élémentaires d'un processus outillé permettant de résoudre le problème de conception sûre et optimale pour le scénario d'aide à la maintenance avionique d'un avion de type Business Jet. Pour cela, nous définirons dans un premier temps un outil

de génération du modèle à partir de bases de données utilisateurs, c'est-à-dire un outil de traduction et transformation des données formelles que nous avons exprimées précédemment en des données de type SPC MDP. Puis nous étudierons la question de l'acquisition des données utilisateurs, au travers de la connexion à des bases de données existantes, de l'utilisation des documents techniques ainsi qu'au travers de la capture de la connaissance métier. Enfin, nous utiliserons ces informations pour évaluer à la fois le modèle SPC MDP et le prototype de processus outillé que nous avons développé, vis à vis de plusieurs critères que nous définirons.

V.1 Construction du processus outillé

Nous avions défini dans l'un des chapitres précédents un ensemble de données formelles sur le scénario du Business Jet. Cependant, le solveur que nous souhaitons utiliser repose sur des données représentant un SPC MDP.

Comme évoqué précédemment, nous devons modéliser trois aspects en SPC MDP :

1. La **fonction de transition** du Processus Décisionnel Markovien, exprimant le modèle physique et les actions possibles,
2. la **fonction de récompense**, portant les objectifs,
3. les **contraintes** en Logique Temporelle (PCTL).

Cette modélisation est complexe en ce que les données d'un SPC MDP ne sont pas facilement manipulables par l'utilisateur : l'expression de la fonction de transition est particulièrement complexe, puisqu'elle consiste à lister toutes les combinaisons états-actions, et à leur associer des états futurs probables ; ceci nécessite dans un premier temps de trouver tous les états, ce qui est déjà en soi une tâche ardue.

Pour cette raison, il n'est pas envisageable en pratique de demander à un utilisateur d'exprimer directement la fonction de transition, la fonction de récompense ou les contraintes PCTL. Ceci implique de choisir un format plus adapté pour exprimer les données formelles. Selon le format choisi, il nous faudra donc effectuer une conversion de notre modèle de données formelles en un modèle SPC MDP.

Dans le processus outillé complet, ceci correspond à une brique de traduction entre deux modèles : d'une part les données formelles, représentées par un format choisi ou potentiellement par plusieurs formats qui se retrouvent combinés par plusieurs étapes du processus outillé ; d'autre part le modèle SPC MDP, exprimé sous son propre format et permettant l'utilisation du solveur. Notons que la réalisation de cette brique de traduction est une part importante du processus outillé : il est nécessaire d'assurer la traçabilité des données tout au long du processus, ce qui peut par exemple être possible en produisant des documents ou modèles intermédiaires.

Pour réaliser cette brique de traduction, nous sommes parti de la fin du processus. Nous avons ainsi étudié deux modèles de données permettant de représenter un modèle SPC MDP : PPDDL, langage formel spécifiquement conçu pour les MDP, et C++/UML puisque C++ est le langage utilisé par l'API du solveur que nous avons développé, c'est-à-dire que C++ permet une connexion directe au solveur. Nous ne parlerons pas dans un premier temps du format utilisé pour représenter les données avant l'étape de traduction - cette question sera traitée dans la section suivante.

Cette section détaille ainsi un retour d'expérience sur ce type de modélisation, à travers la comparaison de ces deux options.

V.1.1 Modélisation du problème dans le langage PPDDL

Exprimer ce problème en SPC MDP nécessite une étape de traduction, pouvant être réalisée par un langage déjà utilisé par la communauté comme un langage pivot, effectuant la liaison entre des modèles industriels et des solveurs : nous avons saisi le modèle précédent dans le langage formel **PPDDL (Probabilistic Planning Domain Definition Language [YS04])**, qui est l'un des langages de référence dans la communauté de planification en environnement probabiliste. En particulier, ce langage permet d'établir une description haut-niveau de notre problème sous la forme d'un *domaine*, c'est-à-dire avec des variables abstraites n et m, puis de *l'instancier*, c'est-à-dire de spécifier la valeur des variables abstraites.

Comme nous le voyons sur l'exemple (Algorithme 7 page 102), il est très intuitif d'exprimer un problème en PPDDL : la première ligne *"type"* définit les objets que nous pouvons manipuler, la seconde *"predicates"* définit les *"fluents"* qui représentent l'état du système. Par exemple, pour représenter un avion à Paris avec le système IRS1 en panne, on pourra simplement dire que les fluents "(at Paris)" et "(failed IRS1)" sont vrais. Par défaut, tous les fluents non mentionnés explicitement sont considérés faux.

Certains fluents peuvent être paramétrés par plusieurs objets, ainsi "repairable" représente le fait qu'un système soit réparable à une escale donnée. Des variables paramétrées plus complexes peuvent

être définies, appelées *functions*, comme par exemple "ft" qui représente la durée depuis laquelle un système donné est en panne. Ces variables ont des valeurs entières ou décimales.

PPDDL permet aussi de paramétrer de la manière suivante : une action peut être appliquée si des conditions, définies dans le champ "precondition", sont remplies ; appliquer une action produit des effets, potentiellement probabilistes, qui sont d'ajouter ou retirer des fluents, ainsi que de modifier la valeur de fonctions dans l'état suivant. Ainsi, dans l'exemple (Algorithme 7 page 102), l'action "repair" concerne un système s et une escale donnée loc ; elle s'applique lorsque l'avion est à cette escale et que le système est réparable à cet endroit ; son effet est alors de retirer le fluent "failed" du système s et de diminuer la fonction "reward" de 1. "reward" est une fonction particulière, définie directement dans le langage et correspondant à la récompense pour les MDP.

Algorithm 7: PPDDL domain – extrait simplifié

```
(:types location system)
(:predicates
  (at ?loc - location)
  (failed ?s - system)
  (repairable ?s - system ?loc -location)
  (in-flight ?l1 ?l2 - location)
  (true))
(:functions
  (ft ?s - system)
  (next ?l1 ?l2 - location))
(:action repair
  :parameters (?s - system ?loc - location)
  :precondition (and
    (failed ?s)
    (at ?loc)
    (repairable ?s ?loc))
  :effect (and
    (not (failed ?s))
    (decrease reward 1)))
```

Comme nous l'avons déjà évoqué, l'une des plus-values principale de PPDDL est de pouvoir définir séparément une version générique, appelée domaine, et un problème instancié correspondant. L'exemple (Algorithme 8 page 103) montre par exemple une instance du problème du business jet correspondant à un plan de vol Paris-Berlin-Nice-Athènes et 3 systèmes pouvant tomber en panne. Le champ "init" désigne tous les fluents étant vrais à l'état initial.

Le champ "pctl" est l'un des ajouts que nous avons effectué au langage PPDDL et permet d'exprimer des contraintes PCTL [HJ94] : "(:pctl (f) (g) p)" correspond à une contrainte $fU_p^\infty g$, signifiant que tous les chemins possibles doivent garantir avec une probabilité $p \in \{0; 1\}$ que la formule booléenne f reste vraie jusqu'à ce que g devienne vraie. Dans cet exemple, la contrainte PCTL représente ainsi le respect de la MEL : les pannes doivent être réparées avant une certaine durée fixe.

La flexibilité de ce langage formel nous a permis de produire rapidement plusieurs séries de tests avec de nombreux paramètres (Algorithme 16 page 212). Les résultats sont satisfaisants en termes de qualité de la solution, et permettent de valider notre approche. PPDDL étant un langage largement supporté par la plupart des planificateurs en environnement probabiliste, il a été très facile de l'utiliser pour représenter notre problème et de le tester immédiatement avec des planificateurs performants. De la même manière, il a été très facile de connecter le langage PPDDL à notre solveur, en s'inspirant des solveurs existant à l'Onera.

Cependant, le retour d'expérience que nous pouvons apporter sur la modélisation PPDDL est que le langage manque de flexibilité vis-à-vis de la connexion à des bases de données extérieures : utiliser des coûts de réparations différents, des temps de réparation différents, ainsi que des probabilités de panne propres à chaque système a immédiatement nécessité de générer le code PPDDL automatiquement à partir de bases de données. Ainsi, l'extrait de code que nous avons présenté précédemment profite par exemple du paramétrage des actions PPDDL, alors que pour le code généré nous avons dû produire

Algorithm 8: PPDDL problem

```
(define (problem business-jet-3)
  (:domain business-jet)
  (:objects Paris Berlin Nice Athene - location
    s1 s2 s3 - system)
  (:init (at Paris)
    (= (next Paris Berlin) 2)
    (= (next Berlin Nice) 1)
    (= (next Nice Athene) 1)
    (= (next Athene Paris) 1)
    (repairable s1 Paris)  (repairable s1 Nice)
    (repairable s2 Paris)  (repairable s2 Nice)
    (repairable s3 Paris)  (repairable s3 Nice)
    (= (ft s1) 0) (= (ft s2) 0) (= (ft s3) 0)
    (true))
  (:pctl (true) (or (> (ft s1) 2)
    (> (ft s2) 2) (> (ft s3) 2)) 0)
  (:metric maximize (reward)))
```

manuellement un nouveau type d'action de réparation par coût différent de réparation.

Le bilan est donc mitigé, comme résumé dans le tableau (Tableau 2 page 104) : d'une part l'effort initial de modélisation est très faible, puisque PPDDL permet de s'affranchir d'une définition manuelle de la fonction de transition d'un MDP ; mais d'autre part l'ajout d'informations propres à certaines instance des objets, c'est-à-dire des paramètres propres à certaines réparations ou certaines escales, semble aller à l'encontre de la philosophie du langage et nécessite de générer du code spécifique. Enfin, certains aspects plus poussés de la modélisation du problème ne pourront vraisemblablement pas être supportés par PPDDL, sans de grandes modifications de la sémantique du langage. Par exemple la paramétrisation des actions par des lois de probabilité de panne, afin de pouvoir spécialiser les actions selon les composants, nécessiterait de générer automatiquement un trop grand nombre d'actions spécifiques.

V.1.2 Modélisation du problème dans les langages UML et C++

La seconde option de modélisation que nous avons explorée est celle d'une représentation du problème directement dans l'API native du solveur PCMDP utilisé, écrit en langage C++. Une modélisation directe correspond à définir explicitement l'espace d'états, l'espace d'actions, la fonction de récompense et la fonction de transition associée. Comme nous l'avons évoqué précédemment, une telle définition est complexe et difficile à maintenir.

Pour alléger ces défauts, nous nous sommes orientés vers une modélisation en deux temps :

1. Représenter le problème sous une forme haut-niveau dans le langage UML (Figure 40 page 206), permettant de mettre en avant les différentes bases de données, les formats d'échanges avec celles-ci basés sur des schémas XML, ainsi qu'une définition générique des actions de réparations possibles ;
2. Traduire ces diagrammes UML en classe C++, permettant de générer à la volée les états et probabilités de transitions du MDP.

Cette approche permet donc une maintenabilité accrue (i.e. capacité au changement de paramètres) et une robustesse aux changements de modèles, puisqu'il suffit de modifier les données haut niveau (UML) pour en voir directement les impacts sur l'implémentation bas niveau (C++). La traduction UML/C++ n'a cependant pas été effectuée automatiquement, puisqu'il nous a fallu utiliser l'API spécifique au solveur.

À nouveau, le bilan est mitigé, comme le montre le tableau (Tableau 2 page 104) : le principal point positif est que nous avons pu ainsi modéliser avec beaucoup de détails notre problème, en prenant en compte des fonctions de coût complexes et des cas particuliers associés à certaines escales. Bien que

l'effort de modélisation initial soit plus élevé, une modélisation native en C++ permet immédiatement d'utiliser le solveur dans des cas réels d'application, avec des bases de données réelles, ce qui permet de valider plus efficacement notre concept d'outil d'aide à la décision ; à l'inverse, la modélisation en PPDDL aurait nécessité une étape supplémentaire de réflexion sur l'échange d'informations à partir de bases de données.

Néanmoins, même avec la vision haut niveau UML, le code natif bas-niveau est très peu lisible par un non initié. Son écriture, et son maintien en cas de changement du modèle, nécessite une connaissance non triviale des MDP et de l'API du solveur : nous avons par exemple dû réécrire une partie de la sémantique de PPDDL en C++ pour pouvoir utiliser des fonctions ressemblant à l'opérateur "forall". Enfin, les deux options de modélisation se sont avérées présenter un léger avantage sur le plan de la performance pour C++ dû à la possibilité d'optimisation de certaines structures de données et l'utilisation de clés de hachage.

Critères	**PPDDL**	**UML & C++**
Lisibilité	+	
Maintenabilité (changement des paramètres)	+	+
Facilité de modélisation	+	
Robustesse aux changement de modèle	+	
Temps de résolution		+
Niveau de détail du modèle		+

TABLE 2 – Techniques de modélisation : PPDDL vs UML/C++

V.1.3 Bilan sur la génération automatique de modèles

Le bilan de cette modélisation est donc que ni PPDDL ni UML/C++ ne sont pleinement satisfaisants pour nos besoins : PPDDL pourrait être qualifié de langage orienté utilisateur, facile à maintenir et concevoir mais limité en expressivité ; alors qu'à l'inverse C++ avec une couche de conception UML serait un langage orienté solveur, expressif et parfaitement adapté à la manipulation des données d'entrée et de sortie, mais illisible pour quelqu'un d'extérieur au domaine.

Comme nous l'avons brièvement évoqué, l'approche finalement adoptée s'est avérée mixte (Figure 18 page 105) : nous avons proposé une séparation des vues utilisateurs et solveur à travers une génération automatique des modèles PPDDL et C++. En concevant des schémas de saisie de type XML ou DSL (Domain Specific Language) au travers desquels un utilisateur peut facilement peupler le modèle de contraintes (MEL) ou les paramètres du modèle physique (plan de vol, logistique,...), il a été possible de pallier les limites de ces deux langages, en générant automatiquement les aspects les plus fastidieux. L'inconvénient principal de cette méthode est son coût de mise en œuvre initial : elle nécessite une réflexion approfondie sur les besoins et la forme des données d'entrée.

Dans le cadre de notre processus outillé, nous avons donc la partie finale du processus (sans avoir pour l'instant la partie initiale) : à partir des données formalisées dans un chapitre précédent, que nous capturons sous la forme de données XML ou DSL, nous pouvons réaliser une génération automatique de modèle PPDDL ou C++ (selon le besoin de traçabilité), permettant la résolution du problème par un solveur reposant sur le modèle SPC MDP et l'algorithme SPC VI.

Dans la section suivante, nous détaillerons les structures de données utilisées comme entrées de ce processus, ainsi que les outils et techniques que nous pouvons utiliser pour faciliter leur acquisition à partir de documents d'ingénierie.

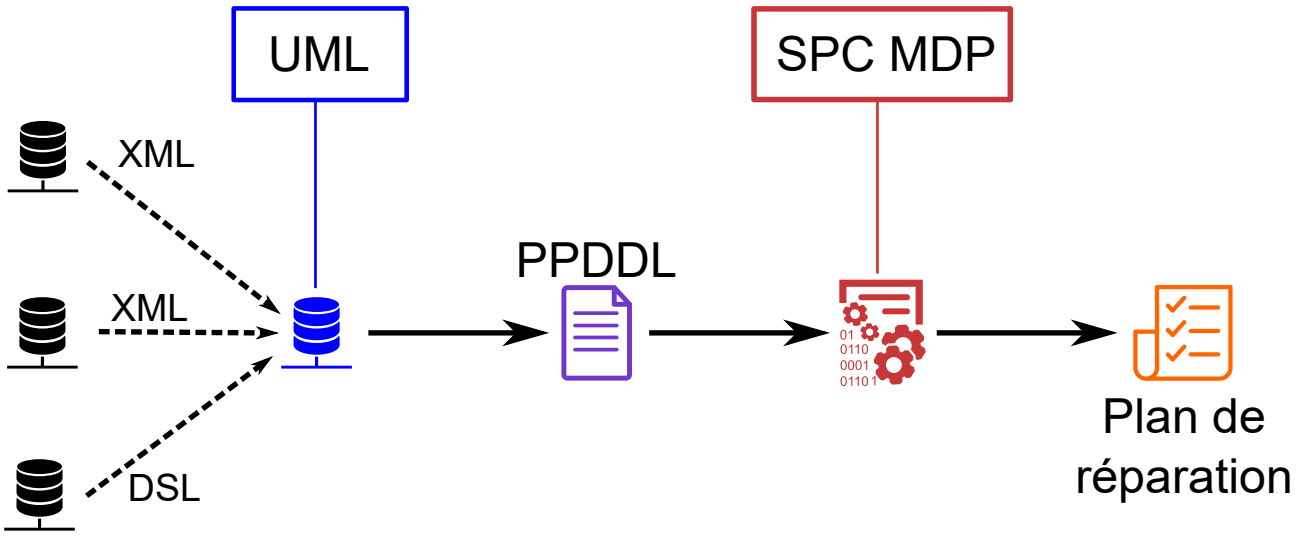


FIGURE 18 – Schéma du processus outillé basé sur une génération automatique de modèles

V.1.4 Capture des documents d'ingénierie pour peupler les structures du modèle

Les données constituant une instance du problème du business jet proviennent de sources multiples :

- Certaines sont disponibles à partir des documents produits par les processus de conception et de certification de l'avion, tels que le Trouble Shooting Manual ou la MEL.
- D'autres sont obtenues à partir de connexions à des bases de données externes au moment de la résolution, telles que le plan de vol, la liste des équipements disponibles ou la liste des équipements déjà en panne dans l'avion ;
- Enfin, certaines données sont destinées à être modifiées au cas par cas, ou améliorées par retour d'expérience, telles que le coût de réparation ou le temps estimé pour chacune des opérations de la procédure.

Pour lier toutes ces données, la toute première étape du processus outillé nécessite donc de capturer plusieurs sources d'origines très différentes. Dans les tests que nous avons réalisés, certaines de ces sources nous étaient accessibles, ce qui nous a permis d'implémenter les différents modules de capture en nous servant de documents réels. Ceci concerne en particulier le Trouble Shooting Manual et la Minimum Equipment List, pour lesquels nous détaillons dans les paragraphes suivants les méthodes de capture utilisées.

D'autres sources ne nous étaient pas accessibles, en particulier concernant les connexions à des bases de données externes ; pour ces sources, nous avons généré un ensemble de données de test et nous avons décrit le type de protocole pouvant être mis en œuvre dans un processus réel pour acquérir les informations manquantes.

Capture du Trouble Shooting Manual

Le Trouble Shooting Manual est un document dont la structure varie pour chaque modèle d'avion. Son rôle principal demeure cependant le même : à partir d'un ensemble de messages de défaillances, il permet de réaliser plusieurs tests interactifs pour isoler les causes racines, puis de conduire les opérations de reconfiguration ou réparation.

Les TSM des modèles d'avion de la génération précédente, tels que les avions A320, sont organisés de la façon suivante :

Organisation standard du Trouble Shooting Manual

- **Table ECAM** : le TSM possède une table d'entrée (ECAM), associant à chaque message ou groupe de messages une référence TSM.
- **Probable Cause** : cette référence TSM présente un texte détaillé expliquant la défaillance, puis propose une liste de causes racines possibles.
- **Fault Confirmation** : elle propose une série de tests (tests BITE le plus souvent) permettant de s'assurer que la défaillance est bien présente. La procédure de chacun de ces tests est potentiellement décrite dans un autre document, l'AMM, présentant des opérations dédiées à la maintenance.
- **Fault isolation** : elle propose ensuite une série d'opérations à effectuer pour lever l'ambiguité restante entre les causes probables. Cette série d'opérations fait aussi référence à des procédures de réparations (AMM) lorsque la cause racine a été identifiée.
- **Test** : enfin, chaque entrée du TSM propose une série de tests à effectuer pour vérifier que la défaillance a bien été résolue.

Nous pouvons immédiatement effectuer deux remarques sur ce document : la première est qu'il est constitué en grande majorité de texte libre, c'est-à-dire que les tests ne peuvent pas être appliqués automatiquement par un ordinateur sans l'assistance d'un opérateur humain, ou sans un formatage supplémentaire des données. La seconde remarque est que le TSM effectue en réalité une partie du travail de diagnostic, puisqu'il considère comme entrée un ensemble de messages de défaillance et non un ensemble de causes racines possibles.

À partir de versions informatiques (pdf) de ce document, nous avons montré qu'il était possible de récupérer un ensemble d'informations nécessaires à la résolution du problème, tel que la liste des causes possibles, les numéros de référence aux procédures de maintenance AMM ainsi que les différents paragraphes décrivant la procédure de test. Ces informations vont en particulier permettre d'effectuer la liaison avec une base de données interne au MCC sur le temps estimé de réparation en fonction des opérations mises en œuvre.

Cependant, cette capture du TSM met en avant de façon flagrante la nécessité d'un format standardisé des procédures de Trouble-Shooting, non-seulement pour assurer la cohérence entre les différentes entrées propres à chaque système, mais aussi pour permettre le traitement automatique d'une partie des procédures.

Ces réflexions nous ont amenés à proposer le format préliminaire ci-dessous (Algorithme 9 page 108), basé sur les arguments suivants :

Concepts principaux de définition d'un format TSM

- L'objectif du TSM est triple : (1) isoler les causes racines responsables, (2) détailler les actions immédiates de reconfiguration et (3) détailler la procédure de remise en état du système.
- Le TSM devrait être entièrement compatible avec des données d'entrée en provenance d'un **système de diagnostic**, c'est-à-dire des données décrivant une liste ordonnée de conjonctions de causes possibles (simple, double, triple pannes ou plus).
- Le TSM devrait **décrire l'impact** de chacune des opérations sur le système, ainsi que l'impact de la procédure globale sur la capacité de l'avion à poursuivre la mission.
- Le TSM devrait détailler des **procédures indépendantes** de l'équipement en place, permettant de conserver une procédure correcte selon toutes les configurations possibles ou selon les évolutions futures envisageables.

Nous proposons un format basé sur une définition XML (Algorithme 9 page 108), qui est le langage d'échange de données le plus populaire dans l'industrie. Le schéma XML associé est constitué d'un ensemble de tâches, correspondant à une sortie possible du diagnostic. Chaque tâche est constituée de trois parties :

- Une entête informative, détaillant les groupes d'ambiguïtés possibles, le matériel nécessaire ainsi que les documents de référence ;
- Une partie de Confirmation, détaillant les tests à effectuer pour confirmer la panne ;
- Puis une partie d'Isolation détaillant toutes les procédures possibles permettant de remettre le système en état, ainsi que leurs effets attendus.

Ces procédures ont chacune un effet sur le système et sont associées à un ensemble d'opérations, c'est-à-dire de séquences alternant tests et actions. Les tests correspondent à des successions d'actions devant amener le système dans un état déterminé (message affiché, paramètre à mesurer ou état visuel à établir) ; le test peut être réussi ou échoué, menant à deux opérations différentes. Une action est une interaction élémentaire renvoyant à un signal pouvant être envoyé au système, à une sélection pouvant être effectuée sur l'interface, ou à une opération spécifique à un équipement ou un matériel qui est alors décrite dans un manuel.

Notons que pour raison de localisation (changement de langage) et de maintenabilité, nous recommandons fortement de limiter la quantité de texte libre présent dans le document, pour préférer la présence de numéro d'identification : au lieu d'attribuer une chaîne de caractères comme nom à la tâche, il est préférable de lui attribuer un numéro et de permettre la connexion par les outils à une base de données associant un texte descriptif dans le langage de l'opérateur, voire un schéma ou une représentation plus explicite.

Enfin, notons que ce schéma reprend la structure du document TSM A320, qui est le deuxième avion le plus vendu au monde, derrière le Boeing 737. Ceci est un argument en faveur de ce schéma, puisqu'il respecte dans une certaine mesure les habitudes de travail des opérateurs de maintenance actuels, mais il s'agit aussi d'un argument en défaveur de ce schéma puisqu'il est indéniable qu'une étude plus approfondie du sujet permettrait de construire depuis la base un format de données adapté à un nouveau processus de maintenance, par exemple en partant du principe que le système est autorisé par lui-même à effectuer certaines opérations de la procédure automatiquement.

Le TSM des modèles existants est conçu pour être utilisé et compris par un opérateur humain, ce qui participe à le rendre obsolète dans le contexte actuel où la nouvelle génération d'opérateurs s'attend à disposer d'outils interactifs et assistés par des appareils intelligents. Un nouveau format tel que celui proposé (Algorithme 9 page 108), qui est conçu pour être utilisé et compris par un système informatisé, est donc indispensable pour les produits et processus futurs.

Capture de la Minimum Equipment List

Le second document provenant de la conception du système est la Minimum Equipment List [EASb], décrivant les conditions minimales selon lesquelles l'avion peut décoller, en particulier concernant les équipements pouvant être inopérants au décollage.

L'organisation de cette liste est laissée libre, dans la mesure où elle satisfait un processus de validation par des autorités compétentes ; l'organisation standard est la suivante :

Algorithm 9: Schéma DTD pour la définition d'une version XML du Trouble Shooting Manual compatible avec un processus informatisé

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE TSM [
        <!ELEMENT Task (information,confirmation,isolation)>
        <!ATTLIST Task IDtask ID #REQUIRED>


        <!ELEMENT information (ambiguityGroups,support,reference)>


        <!ELEMENT ambiguityGroups (situation)+>
        <!ELEMENT situation (rootCause)+>
        <!ATTLIST rootCause IDsystem ID #REQUIRED IDfailureMode ID #REQUIRED >

        <!ELEMENT support (supportEquipment)*>
        <!ATTLIST supportEquipment IDequipment ID #REQUIRED quantity CDATA #REQUIRED>

        <!ELEMENT reference (referenceDocument)*>
        <!ATTLIST referenceDocument IDdocument ID #REQUIRED IDentry ID #REQUIRED>


        <!ELEMENT confirmation (procedure)*>
        <!ELEMENT procedure (operation*,effect)>
        <!ELEMENT operation (test*,actionIfOK,actionIfKO)>

        <!ELEMENT test (action*,observation*)>
        <!ELEMENT actionIfOK ((operation+ | action*),effect)>
        <!ELEMENT actionIfKO ((operation+ | action*),effect)>

        <!ELEMENT effect (observation)*>

        <!ATTLIST action IDSsystem ID #REQUIRED IDaction ID #REQUIRED>
        <!ATTLIST observation IDobservation ID #REQUIRED IDvalue ID #REQUIRED>

        <!ELEMENT isolation (procedure)*>
]>
```

Organisation standard de la Minimum Equipment List

- La MEL est répartie en chapitre selon les catégories de système (ATA).
- Chaque chapitre est séparé selon les numéros d'équipements contenus dans ce système.
- Pour chacun de ces équipements, la MEL précise le nombre d'éléments installés, le nombre d'éléments nécessaires au dispatch, le temps (A,B, C ou D) maximal possible avant une réparation suivi par un texte libre détaillant des conditions supplémentaires.
- Le texte libre répète sous forme textuelle le nombre d'équipements pouvant être inopérants, sous un ensemble de conditions listées ; ces conditions portent sur des vérifications de fonctionnement qu'il est nécessaire d'effectuer, telles que le fonctionnement de la chaîne de redondance.

De façon similaire au TSM, la capture de ce document met en avant l'inadéquation de la MEL actuelle aux systèmes modernes : face à une complexité croissante, il est nécessaire de concevoir un nouveau format de MEL adapté à la gestion de cette complexité, à défaut d'une approche nouvelle du processus d'évaluation du dispatch.

Nous proposons le format préliminaire ci dessous (Algorithme 10 page 110), basé sur la structure existante et les recommandations suivantes :

Concept principaux de définition d'un format MEL

- La MEL devrait être basée en entrée sur une **liste de capacités**, et non sur l'état du système qui est potentiellement non disponible durant une escale.
- La MEL devrait détailler des conditions **indépendantes de l'équipement** en place, permettant de conserver la cohérence de la MEL dans le cas de changement de configurations.
- La MEL devrait faire explicitement **référence à une procédure de test**, potentiellement automatisable, permettant d'évaluer si la condition de dispatch est remplie.

Le schéma XML que nous proposons (Algorithme 10 page 110) respecte donc la structure d'origine, en minimisant la quantité de texte libre. Il est constitué d'un ensemble d'entrées, faisant chacune référence à une capacité ; une capacité n'est pas l'état d'un équipement, mais une information observable sur un service rendu par cet équipement, par exemple par l'intermédiaire de messages de fonctionnement dégradé ou de défaillance. Chaque entrée est structurée selon les différents états possibles de cette capacité, par exemple différents niveau de défaillance ; pour chacun de ces états possibles, la MEL propose plusieurs situations GO-IF sous lesquelles l'avion est autorisé à décoller. Une condition GO-IF contient un ensemble de tests devant tous être passés pour permettre le dispatch ; ces tests font éventuellement référence à un résultat de diagnostic où à l'état d'autres capacités. Puis chaque condition GO-IF référence une procédure qu'il est nécessaire d'effectuer immédiatement, suivie enfin d'un temps maximal durant lequel l'avion est autorisé à voler avec cette capacité dégradée. À titre informatif, la condition GO-IF pourrait aussi présenter la liste des équipements concernés par la condition de dispatch, ainsi que le nombre d'éléments devant être opérants ; cependant, ces informations sont déjà présents sous une certaine forme dans les tests à effectuer, nous ne les avons donc pas inclus dans la définition du format de données.

Notons que la structure a ici été modifiée en profondeur : là où l'ancien format de MEL permettait d'ignorer certaines défaillances d'équipements sous certaines conditions, cette nouvelle structure permet en vérité d'ignorer certaines pertes de capacité (c'est-à-dire certains messages de défaillance) sous certaines conditions. La distinction est importante sur le plan d'un processus automatisé, puisqu'il doit permettre la connexion avec un maximum de systèmes informatiques présents dans l'avion ; mais elle a comme inconvénient principal d'être moins intuitive pour un opérateur humain, qui a pour

Algorithm 10: Schéma DTD pour la définition d'une version XML de la Minimum Equipment List compatible avec un processus informatisé.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE MEL [
        <!ELEMENT Entry (capacityStatus+)>
        <!ATTLIST Entry IDcapacity ID #REQUIRED>


        <!ELEMENT capacityStatus (GOIF*)>
        <!ATTLIST capacityStatus IDcapacityStatus ID #REQUIRED>


        <!ELEMENT GOIF (test*,procedure*, time)>
        <!ATTLIST time maximalTime CDATA #REQUIRED>

        <!ELEMENT procedure (operation*,effect)>
        <!ELEMENT operation (test*,actionIfOK,actionIfKO)>

        <!ELEMENT test (action*,observation*)>
        <!ELEMENT actionIfOK ((operation+ | action*),effect)>
        <!ELEMENT actionIfKO ((operation+ | action*),effect)>

        <!ELEMENT effect (observation)*>

        <!ATTLIST action IDSsystem ID #REQUIRED IDaction ID #REQUIRED>
        <!ATTLIST observation IDobservation ID #REQUIRED IDvalue ID #REQUIRED>

]>
```

l'instant l'habitude de raisonner en termes d'équipement en panne et d'équipement à réparer. Bien que cette catégorisation en équipements ne nous semble plus représentative des architectures avioniques modernes, il sera probablement nécessaire de permettre une vue de la MEL adaptée à la communication avec des opérateurs humains ; une telle vue utiliserait le format de données défini précédemment pour permettre à un opérateur d'explorer la MEL selon des critères d'entrée multiples, par exemple à partir d'une liste d'équipements défaillants.

Capture des coûts à partir de préférences

La capture des deux documents précédents, TSM et MEL, nous permet en pratique d'obtenir le modèle du monde physique (liste des systèmes, effets des défaillances, logistiques et pièces nécessaires,...) ainsi que le modèle des contraintes (MEL, Interval repair). Il ne permet pas en revanche d'obtenir la totalité du modèle des actions contrôlables, en particulier le modèle de coût.

Ce coût peut dans un premier temps être un coût purement financier : pour chaque action de réparation, une base de données nous permet d'évaluer le coût de mise à disposition des ressources à chaque escale, nous donnant le coût total de l'action. Bien que nous n'ayons pas eu accès à une telle base de données dans notre étude, la mise en place d'une telle base ne présente pas de défis nécessitant une étude particulière, sinon une étude de marché et d'accords de coopération pour l'acheminement des pièces de rechange.

Néanmoins, le calcul du coût devient complexe dès lors qu'on souhaite prendre en compte d'autres paramètres : par exemple, un utilisateur pourrait vouloir favoriser une réparation rapide d'un équipement, ou encore permettre un retard au décollage avec une certaine pénalité financière. Ces paramètres sont à la fois complexes à déterminer sur le plan économique et complexes à mettre en relation avec le coût de la réparation.

La méthode naïve d'agrégation des coûts est une agrégation pondérée : pour calculer le coût d'une action de réparation, on peut par exemple faire la somme du coût des pièces nécessaires multipliée par un certain facteur et du coût de la pénalité de retard multiplié par un second facteur. Si la pénalité représente un coût financier, alors il est logique de fixer les deux facteurs à 1 ; s'il s'agit d'un coût qui n'a pas de sens monétaire, il faut en revanche se poser la question de ce que l'utilisateur préfère : s'il a une aversion importante pour le retard, la pénalité de retard aura un poids important ; à l'inverse si on souhaite minimiser l'investissement de la part du MCC on fixera un poids plus important sur le coût d'acheminement.

Plus précisément, on peut envisager les paramètres de coût suivants :

1. le coût d'utilisation des équipements,
2. la pénalité de retard au décollage,
3. la pénalité d'abandon de mission,
4. la pénalité de déviation (comprenant le coût du fuel et le retard),
5. un coût fictif associé au temps avant réparation,
6. un coût fictif associé à la disponibilité de l'appareil, par exemple en cas de départ anticipé d'une escale.

Cette notion est celle de la **préférence** et est fondamentale en aide à la décision multi-critères [FGE05]. De nombreux critères d'agrégation existent, permettant d'exprimer plus de nuances dans la préférence que l'agrégation pondérée ; [GL10] et [Bel86] présentent par exemple plusieurs approches de modélisation de la préférence au travers de nouveaux opérateurs, tandis que des travaux tels que [CdA07] utilisent une modélisation de nouveaux paramètres pour améliorer la décision, telle que la modélisation de l'incertitude au travers de différentes catégories avec application à la maintenance préventive à partir de données partielles. Les travaux de [LLH05] sont tout aussi intéressants, en ce qu'ils permettent d'ajuster les critères de préférence à partir d'une phase d'apprentissage : en demandant à l'utilisateur de répondre à une série de questions telles que des comparaisons entre deux situations, il est possible de déterminer un certain nombre de paramètres pour l'opérateur d'agrégation.

Dans les exemples que nous avons traités pour cette étude, tous les coûts ont été générés aléatoirement et agrégés à partir d'un critère pondéré. Cependant, l'état de l'art minimal que nous avons

construit précédemment montre au premier abord l'intérêt des techniques d'apprentissage dans le processus d'aide à la décision global. Le protocole de capture des préférences de l'utilisateur envisageable pour le processus complet est donc le suivant :

Protocole de capture des préférences

1. On **génère aléatoirement des scénarios** de réparation plausibles, en détaillant pour chacun le coût pour le MCC, le coût pour le propriétaire de l'avion, le nombre de minutes de retard, si la mission est complétée ou non, avec ou sans déviation, ainsi que le temps envisagé avant réparation et le temps d'immobilisation de l'appareil avant qu'il puisse repartir.
2. Muni de ces paramètres, on interroge un ou plusieurs **experts du domaine** pour capturer leurs préférences, au travers de questions ciblées sur ces scénarios.
3. On réalise un **apprentissage** des réponses de ces experts, permettant de déterminer les paramètres de préférence pour la fonction d'agrégation des coûts.

Cette méthode à l'avantage de capturer une agrégation des coûts à partir de la connaissance du domaine, obtenue par l'interrogation d'experts ; elle permet d'obtenir des décisions cohérentes avec la stratégie actuelle du MCC, en particulier concernant sa relation client, et autorise le MCC à modifier le coût ultérieurement, par exemple au travers de retours d'expérience. Cette méthode à cependant les deux inconvénients suivants : (1) elle nécessite de disposer d'experts du domaine et (2) elle ne prend pas en compte la précision estimée des paramètres de coût. En effet, en réalisant une agrégation des différents paramètres de coût, nous agrégeons aussi dans une certaine mesure les imprécisions sur les valeurs obtenues : le coût d'utilisation des équipements peut en réalité varier de quelques centaines d'euros, ce qui par exemple avec une agrégation par pondération donnera une erreur sur le coût total de 100 multiplié par le facteur de pondération. C'est alors d'autant plus critique lorsque la marge d'erreur est grande, ou que le paramètre de coût se retrouve avec une forte importance dans la préférence. Par exemple, un critère tel que la déviation aura vraisemblablement un impact très important sur le coût global, puisque les solutions présentant une déviation seront vraisemblablement considérées en dernier, mais aura au même moment une grande marge d'erreur, puisqu'il est difficile de chiffrer a priori le coût monétaire et le préjudice de réputation associés à un retard aussi important.

Cette marge d'erreur dans le coût cumulé global signifie que trois approches sont possibles :

1. **On ne considère que les paramètres de coût précis.** Ceci comprend donc uniquement les coûts liés à l'utilisation, et éventuellement au retard de mission. Le calcul d'un coût global est alors très simple, ne nécessitant pas de fonction d'agrégation complexe ; la solution optimale obtenue est garantie comme étant la meilleure pour les critères choisis, mais elle peut potentiellement ne pas être satisfaisante selon les préférences (cachées) de l'utilisateur.
2. **On propose une solution ainsi que l'évaluation de l'impact de la marge d'erreur.** Ceci implique de pouvoir calculer dans quelle mesure une erreur sur le coût ou une erreur sur les préférences impacte la solution optimale ou le coût de la solution optimale. Couplée à un système permettant à l'utilisateur de modifier ses préférences, ou d'interroger la valeur estimée d'un plan de réparation particulier, cette solution permet à l'utilisateur de se représenter dans quelle mesure la solution optimale est réellement celle qui l'intéresse.
3. **On propose plusieurs solutions, prenant en compte des marges d'erreur.** Ceci implique que l'utilisateur devra choisir l'une de ces solutions en fonction de son expérience, ou construire une solution qui correspond au mieux au cadre délimité par les solutions proposées. Cette solution décharge la responsabilité d'une erreur sur l'utilisateur, mais lui impose donc une charge supplémentaire de décision.

Dans les produits d'aide à la décision existants, la première solution correspondrait par exemple à un système GPS qui ne proposerait qu'un seul chemin menant à la destination - celui ayant la distance

la plus courte ; la seconde option proposerait une estimation du temps de trajet en fonction du trajet avec une marge d'erreur, permettant à un utilisateur de questionner le système sur un autre trajet pour comparer les estimations. La troisième option proposerait par exemple trois routes : l'une minimisant la distance, l'une minimisant le temps avec embouteillages, l'une minimisant le coût des péages.

Comme pour un système GPS, les trois méthodes ont des inconvénients : la première option peut proposer un chemin particulièrement chargé, causant une perte de temps conséquente ; dans la seconde option, l'utilisateur peut être amené à tester de nombreux chemins avant d'en trouver un qui lui semble raisonnable ; enfin dans la dernière option l'utilisateur a la responsabilité de la décision, ce qui implique qu'il doit évaluer avec son expérience dans quelle mesure la solution minimisant la distance est préférable à la solution minimisant le temps d'embouteillage.

Le choix d'une de ces options dépend ainsi en grande partie du contexte économique dans lequel se place le système d'aide à la décision : une entreprise disposant d'experts du domaine aura un avantage net à choisir la troisième option, tandis qu'une entreprise disposant d'une bonne évaluation de la marge d'erreur aura un avantage à choisir la seconde ; la première option enfin est celle permettant le plus de prédictibilité, par exemple lorsque l'entreprise est tenue contractuellement à un certain niveau de performance. Des solutions regroupant plusieurs de ces options peuvent évidemment être envisagées.

V.1.5 Développement d'un démonstrateur de saisie des paramètres utilisateurs

La dernière pièce manquante pour construire le processus global a été de réaliser un environnement graphique accueillant tous les éléments décrits précédemment (Figure 19 page 114). Cet environnement permet la connexion à de multiples données entrantes, générées aléatoirement pour les besoins de l'exemple selon les formats décrits dans les paragraphes ci-dessus.

Il propose alors dans une *première étape* une visualisation des données à l'utilisateur sous la forme de deux éléments :

- **Un plan de vol**, décrivant un contexte géographique sur le chemin parcouru par l'avion, le chemin restant, l'objectif de mission ainsi que les déviations possibles.
- **Un historique des évènements**, décrivant un contexte temporel sur les évènements passés, les évènements prévus ainsi que les contraintes temporelles pour chacune des escales.

L'élément essentiel de ces deux vues, à ce stade du processus, est de proposer à l'utilisateur une évaluation synthétique de l'urgence de la décision. Ceci est primordial dans le cadre d'un MCC ou un même opérateur peut-être amené à gérer plusieurs tâches en parallèle ; il doit donc disposer rapidement des informations lui permettant d'attribuer des priorités et d'organiser les problèmes à traiter.

La *seconde étape* est une étape de pré-traitement des données : après avoir récupéré l'ensemble des alertes et observations possibles à partir du lien datalink, la connexion à un outil de diagnostic nous permet d'évaluer dans quelle mesure la situation est connue, sa criticité en termes de dispatch ainsi que les procédures (donc les équipements nécessaires et temps de réparation requis) à effectuer. Enfin, l'outil génère les modèles intermédiaires PPDDL ou C correspondant aux différents critères de préférence choisis, qui sont lus et résolus par le solveur SPC MDP.

Les résultats du solveur SPC MDP sont interprétés dans la *troisième étape* en termes d'actions à effectuer dans chaque escale et de moyens à réserver. L'utilisateur peut alors choisir d'effectuer une nouvelle résolution avec des paramètres différents, ou envoyer la solution choisie aux différents opérateurs des sites concernés afin de préparer l'arrivée de l'avion.

L'élément d'ergonomie important pour cette dernière étape est l'interactivité : plutôt que de proposer une seule solution à l'utilisateur, il est indispensable de fournir des explications sur le choix de cette solution, par exemple en faisant référence aux critères de préférence ; l'utilisateur peut ainsi évaluer l'impact de ce plan de réparation sur le système ainsi que la proportion en coût de chacun des critères ; il peut alors demander une nouvelle résolution, selon des nouveaux paramètres, ou bien proposer directement une version modifiée de la solution initiale. En son cœur, l'outil d'aide à la décision doit donc proposer un mécanisme permettant de jouer facilement des scénarios : l'utilisateur doit pouvoir proposer un plan de réparation puis demander au système de l'évaluer, en comparaison au plan optimal, ou de le modifier automatiquement.

Notons que bien que cette fonction ait été envisagée pour le démonstrateur, elle n'a pas été implémentée, pour la raison principale que son apport n'aurait pas pu être évalué rigoureusement sans un ensemble de données réelles et une personne jouant le rôle de l'opérateur de MCC.

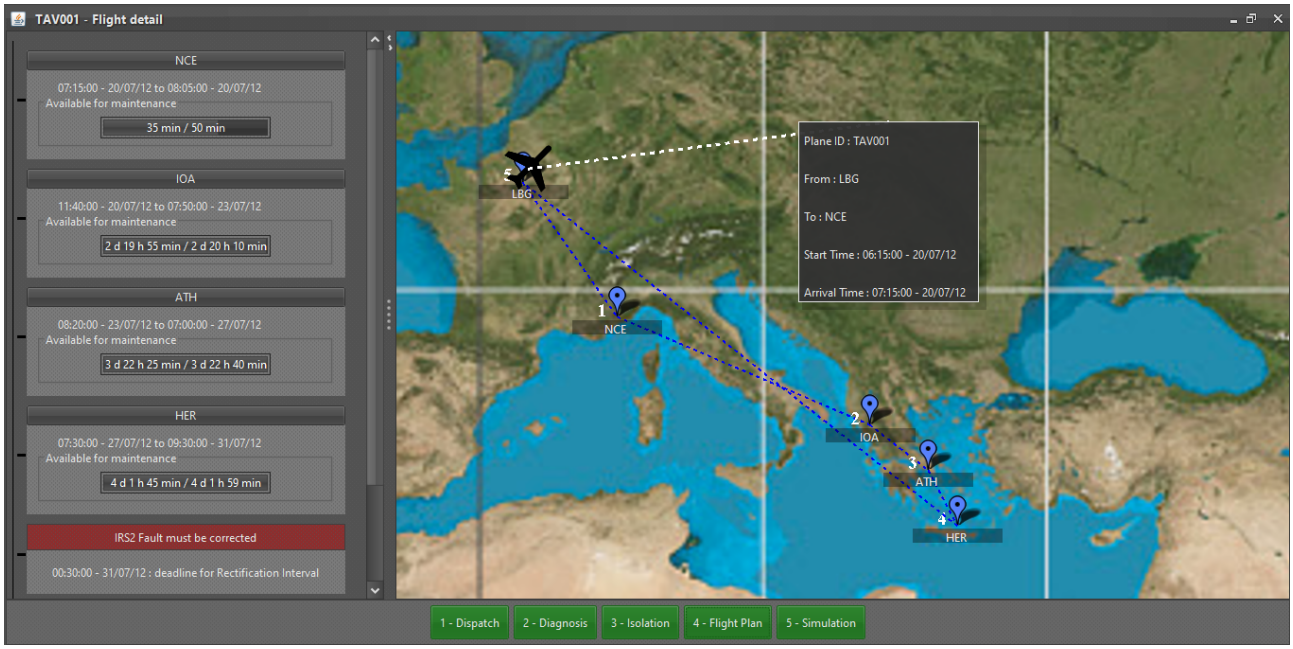


FIGURE 19 – Capture d'écran du démonstrateur portant le processus outillé complet

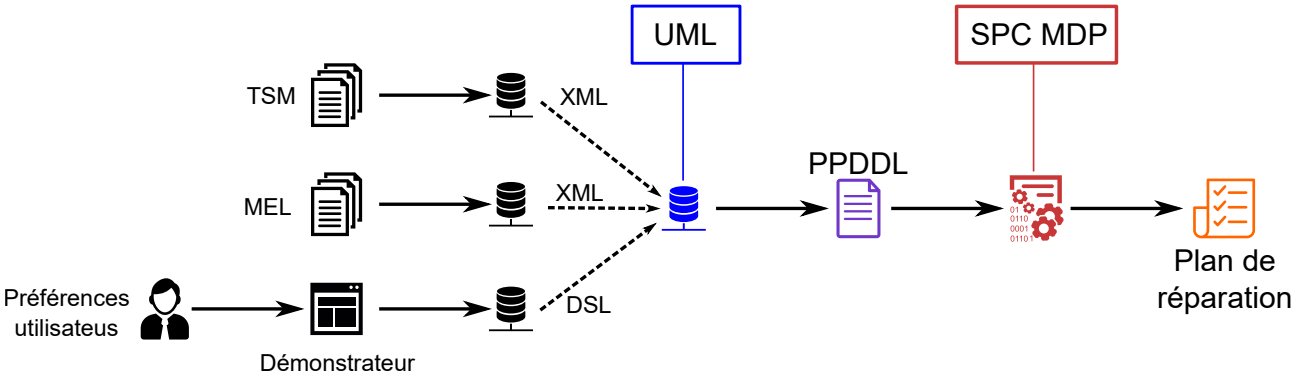


FIGURE 20 – Schéma du processus outillé complet

V.2 Évaluation de la synthèse automatique de plan de réparation

Dans les paragraphes précédents, nous avons établi les différentes étapes nécessaires pour obtenir un outil d'aide à la décision pour la maintenance avionique (Figure 20 page 114). Cet outil repose sur la capture de différentes données pour proposer des plans de réparations possibles à l'opérateur de maintenance, prenant en compte à la fois les aspects de logistiques, de sécurité, ainsi que les résultats de diagnostic et de pronostic.

L'objectif lors de la création de ce processus outillé était de pallier des déficiences des méthodes existantes, en particulier sur les points suivants :

- Elles ne permettent pas de traiter l'ensemble des données existantes en même temps.
- Elles n'offrent pas des garanties suffisantes à la fois sur l'optimalité et sur la maîtrise des risques.
- Elles sont potentiellement longues, à la fois au niveau de la mise en place du processus que dans le temps nécessaire pour la prise de décision.

Pour évaluer ce processus outillé, nous allons ainsi nous baser sur les critères suivants :

- Le **temps de mise en œuvre** de la décision, qui doit être négligeable par rapport à une fenêtre de décision de 30 min correspondant au temps classique disponible à une escale.
- **L'utilisabilité**, c'est-à-dire la qualité de la décision obtenue et la facilité à l'utiliser dans des conditions réelles. Le processus outillé doit ainsi nécessiter une formation minimale, lui permettant d'être intégré dans les processus existants et d'être utilisé par des opérateurs de maintenance avec peu de difficulté de transition.
- La **faisabilité**, c'est-à-dire la facilité de mise en place du processus. Cet aspect comprend à la fois le temps de capture des données nécessaires à la résolution, leur accessibilité ainsi que la facilité de mise en place de l'équipement dédié nécessaire au bon fonctionnement de l'outil.

V.2.1 Évaluation des performances de la résolution en temps de calcul

La principale limitation concerne le temps de calcul : pour ce type de problèmes, le temps de résolution explose de manière exponentielle en fonction de la taille des données d'entrée. La figure (Figure 21 page 116) montre que le facteur déterminant est bien le nombre N de systèmes (croissance exponentielle), et non la taille du plan de vol (croissance linéaire). Ainsi, pour 6 systèmes pouvant tomber en panne le solveur explore par exemple plus d'un million d'états et trouve une solution optimale et valide en 32 secondes.

Deux pistes d'amélioration peuvent être envisagées : réduire de manière pertinente le nombre de systèmes en entrée ; ou trouver un solveur plus rapide, voire non-optimal. Un tel solveur pourrait en particulier profiter de la structure très particulière du problème : notre modèle présente très peu de boucles, et s'apparente dans presque tous les cas à un GSSP (Kolobov et al.2012), qui peut être résolu à la volée de manière bien plus rapide. Une résolution à la volée permet de n'explorer que les états qui semblent les plus intéressants, réduisant de manière importante le nombre d'états visités. Comme nous le verrons dans la seconde partie de cette étude, un algorithme construit autour de ce principe peut être construit et appliqué à ce problème.

Toutefois, la piste principale d'évolution concerne la réduction des données d'entrée : lorsqu'une panne d'un instrument de localisation survient, il n'est par exemple pas pertinent de considérer une panne possible de la ventilation ; il est donc possible d'effectuer un tri en amont des systèmes pouvant impacter la réparation de la panne venant de survenir. Ce tri peut concerner par exemple les critères suivants :

- Considérer les systèmes faisant partie de la chaîne de redondance et/ou mentionnés dans la MEL.
- Considérer les systèmes pouvant prochainement tomber en panne (pronostic).
- Considérer les systèmes dont une panne potentielle serait critique et pourrait retarder ou empêcher une réparation à temps.

Nous avions déjà évoqué dans un chapitre précédent la pertinence de proposer deux options de résolution - une rapide et une plus longue. Ceci correspond à deux besoins différents de l'utilisateur, selon qu'il doit obtenir une réponse préliminaire rapide ou obtenir une réponse détaillant toutes les implications de la solution choisie. Nous avions chiffré lors d'un dimensionnement grossier que le

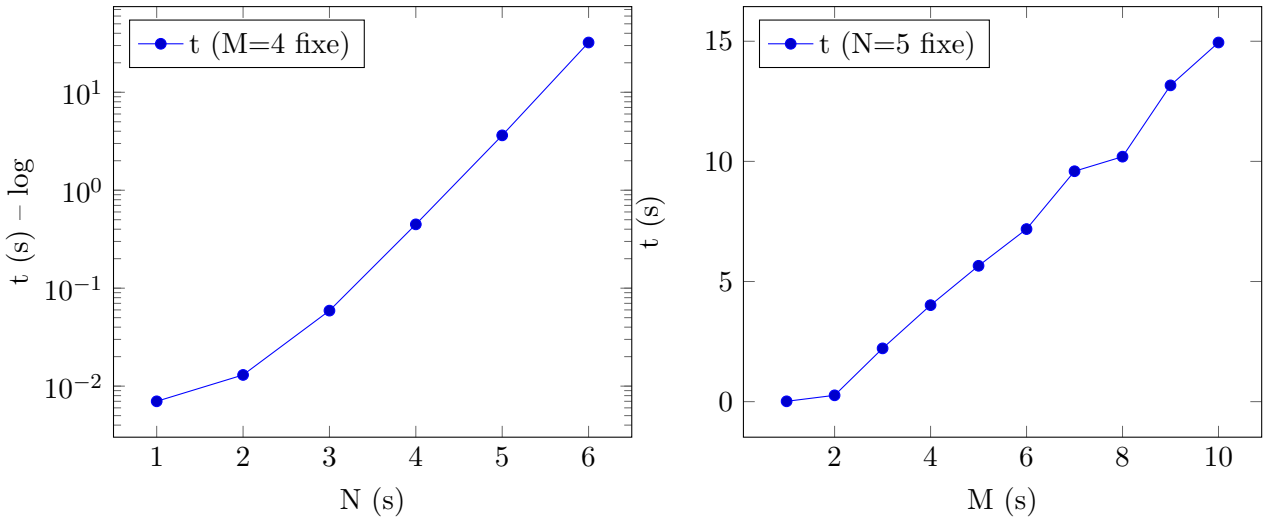


FIGURE 21 – Comparaison des temps de résolution selon N (nombre des systèmes) et M (nombres d'escales). La courbe N est en échelle log.

nombre de systèmes concernés pouvait être réduit à un ordre de grandeur de 10 pour la résolution rapide et à un ordre de grandeur de 100 pour la résolution complète.

Nos résultats montrent donc que cette méthode est applicable dans un contexte industriel pour une résolution rapide, puisque le temps de résolution est de l'ordre de la minute. Cependant, en raison de la croissance exponentielle du temps de résolution en fonction du nombre de systèmes, cette méthode n'est pas envisageable pour une résolution complète : les courbes obtenues (Figure 21 page 116) montrent que l'ordre de grandeur du temps de résolution est de $M * 10^{N-5}$ secondes, ce qui fixe $N = 9$ comme le maximum raisonnable de systèmes pouvant être traités dans le cadre d'un MCC (2 heures et 50 minutes) et $N = 11$ le maximum raisonnable de systèmes pouvant être traité lorsque le temps n'est pas contraint (près de 12 jours).

Nous pouvons étudier trois formes particulières de réduction :

Réduction à un seul système

Le cas extrême est de restreindre la résolution à un système - celui étant défaillant à l'état initial. Cette méthode nous donne le même type de solution qu'une méthode de planification classique, et à l'avantage d'être presque instantanée (10^{-2} secondes dans l'exemple 21).

Une de ces deux conditions est suffisante pour garantir que cette réduction est adaptée :

Conditions suffisantes pour la réduction à un système

1. La solution optimale doit proposer une réparation au temps le plus tôt possible. Ainsi, si une nouvelle panne survient il sera possible de modifier le plan de réparation prévu pour le prendre en compte.
2. Tous les systèmes impactés par la défaillance venant de se produire ont une contrainte MEL plus longue que la durée de la fin de mission.

Avec ces conditions, il est évident que le plan choisi demeure le plan optimal même si la résolution ne prend pas en compte les défaillances pouvant survenir : si une nouvelle défaillance survient, cette défaillance ne provoquera pas d'immobilisation de l'avion, ou en tout cas pas d'immobilisation qu'il aurait été possible d'éviter si on avait choisi un autre plan de réparation.

Réduction à l'ensemble des systèmes impactés ou impactants

L'intuition naturelle est de restreindre les systèmes à ceux immédiatement concernés par la panne venant de survenir. Ceci concerne les systèmes dont la défaillance nécessiterait un changement du plan de réparation, par exemple en cas de non respect des contraintes de temps ou de MEL, ou des systèmes présents dans les conditions de dispatch associés à la panne qui vient de survenir, par exemple les systèmes de la chaîne redondante.

On peut calculer ces systèmes de la manière suivante : à partir de la MEL, on détermine tous les systèmes concernés par les conditions GO-IF des défaillances présentes dans l'avion. On ne garde parmi ces systèmes que ceux nécessitant des opérations complexes, c'est-à-dire nécessitant un temps de réparation non négligeable ou des équipements n'étant pas présents au même prix à toutes les escales. On ajoute à cet ensemble de systèmes tous les systèmes dont les conditions GO-IF dépendent de la défaillance venant de survenir. On définit ainsi l'ensemble des systèmes impactés par la défaillance et des systèmes impactant le maintien des services proposés par l'équipement défaillant.

Il est évident que cet ensemble des systèmes ne suffit pas à garantir que les plans de réparations obtenus sont identiques aux plans optimaux qui seraient obtenus si le solveur prenait en compte tous les systèmes : on peut par exemple imaginer un scénario où une panne d'un autre système survient, nécessitant une réparation immédiate empêchant la réparation initiale ; si la réparation initiale ne peut pas être replanifiée à une escale ultérieure, alors l'avion se retrouve bloqué au sol, ou tout du moins obligé de subir des pénalités de retard.

Ce type de scénario ne peut se produire si l'on conserve la condition (1) que les réparations ont lieu le plus tôt possible. Dans les faits, l'ajout des systèmes impactés/impactant est donc intéressante lorqu'il n'est pas possible d'assurer la condition (2) selon laquelle les systèmes impactés ont une contrainte MEL plus longue que la mission.

La condition (1) peut être prise dès lors qu'il n'y a aucun avantage à repousser la réparation ; ceci est notamment le cas lorsque le coût des pièces de rechange est le même à toutes les escales pour les équipements impliqués dans la réparation. La perte financière maximale résultant du choix de cette option est donc facile à évaluer : il s'agit de la différence entre le coût de la réparation au plus tôt et de la réparation au moins cher.

La condition (1) peut cependant être allégée si au cours de la mission une escale est fixée comme permettant toutes les réparations, à la fois en termes de coût et en termes de temps de réparation ; c'est notamment le cas pour les avions disposant d'un hangar de maintenance avec des visites régulières, ou des missions longues dans laquelle l'avion réalise un arrêt long dans un aéroport, pour lequel les équipements de réparation peuvent être obtenus au plus bas coût. Lors de la résolution, une telle escale agit comme une borne supérieure au temps de réparation de la panne : on sait que si la défaillance ne peut pas être résolue avant cette escale, il sera possible de la résoudre à ce moment là. Sous de telles conditions, la sécurité du vol et la capacité de dispatch ne sont pas mises en cause lorsqu'on ne prend pas en compte tous les systèmes.

Analyses de plusieurs combinaisons de systèmes

La dernière approche de réduction consiste à effectuer plusieurs résolutions avec un faible nombre de systèmes, puis à tirer des conclusions sur ces différentes résolutions sur le plan optimal.

L'avantage de cette approche est que l'algorithme est particulièrement rapide pour une résolution de 1 à 3 systèmes. Il est par exemple possible d'effectuer une centaine de résolutions à 2 systèmes en l'espace de quelques secondes. L'inconvénient principal est que les résolutions multiples ne nous donnent que peu d'information sur la manière de les corréler en une solution cohérente.

Une approche envisageable est par exemple de regrouper les stratégies de réparation à un seul système supplémentaire, et de les comparer à la stratégie obtenue sans système autre que le système défaillant à l'état initial : si toutes les stratégies de réparation du système ayant subi la défaillance initiale sont les mêmes, alors cette stratégie semble robuste à une panne supplémentaire ; si certaines stratégies sont différentes, alors cela veut dire que ces pannes peuvent perturber le plan prévu. En particulier, si certaines de ces stratégies imposent que la réparation ait lieu plus tôt que prévu par la stratégie à un seul système, alors la stratégie optimale devra prendre en compte la panne possible de

ces systèmes.

Plus précisément, en termes de processus décisionnel markovien nous devons comparer entre ces stratégies le chemin d'exécution le plus probable (Most probable path) et détecter parmi les solutions proposées quel est le temps minimal choisi pour la réparation : fixer la réparation plus tard que ce temps est impossible pour des raisons de sécurité ; en revanche, il n'est pas garanti qu'il ne faille pas fixer la réparation plus tôt, puisqu'une combinaison de deux pannes ou plus peut à nouveau perturber le plan de vol.

Il est possible de répéter le même processus (résoudre le problème, trouver le "Most probable path" et trouver le temps de réparation minimal) avec des combinaisons de 2 pannes, puis de 3 pannes, bien que le temps de résolution (et le nombre de combinaisons à étudier) ne grandisse de façon drastique à chaque augmentation. Ceci provoque une erreur en termes de coût vis-à-vis de la politique optimale, puisque toutes les défaillances ne sont pas considérées, qui est au moins aussi grand que la perte maximale causée par les combinaisons de pannes non envisagées, multipliée par la probabilité d'occurrence de chacune de ces combinaison.

On peut par exemple envisager un problème où toutes les situations à 2 pannes ont été anticipées, mais où certaines situations à 3 pannes ont des conséquences importantes en termes de coût, par exemple puisque trois pannes précises mènent à un retard important au décollage ; alors on peut chiffrer le coût de ne pas avoir anticipé cette combinaison de pannes comme étant le produit des probabilités des trois pannes et du coût du retard. On peut se rendre compte que si les trois pannes sont très peu probables la perte financière est acceptable. Pour calculer le coût total de n'avoir anticipé aucune combinaison de 3 pannes, il faudrait faire la somme de ces pertes sur toutes les combinaisons de 3 pannes possibles ; en réalité, les probabilités de défaillance ont souvent plusieurs ordres de grandeur de différence, et il suffit donc le plus souvent de limiter cette somme aux combinaisons les plus probables.

Dans le cas général, en considérant une combinaison de 2 pannes cette perte est minime (en considérant un ordre de grandeur maximal de 10^{-3} pour les probabilités de défaillance des systèmes critiques) et justifie de limiter la recherche à une combinaison de 2 pannes maximum, voire aux combinaisons de 2 pannes les plus probables ($>10^{-6}$).

Bien que cette méthode apporte une réponse satisfaisante et rapide dans une grande majorité des cas, notons qu'il est possible de construire des cas où la solution obtenue n'est pas valide, c'est-à-dire qu'une combinaison extrêmement improbable de défaillances amène à l'immobilisation de l'avion et aurait pu être évitée si une autre décision avait été prise. Il est possible de chiffrer ce risque, puisqu'il s'agit simplement du produit maximal des probabilités des combinaisons non testées ; ce risque peut alors être comparé à un certain seuil, de la même manière que la probabilité de respecter une contrainte est comparée à un seuil dans les contraintes PCTL.

Cette méthode est donc valide lorsqu'il est possible de définir un seuil de probabilité de respect des contraintes et que les contraintes les plus probables sont soit (1) incluses dans les tests de combinaison, soit (2) ont un temps de réparation minime devant le temps disponible aux escales visitées avant la date prévue pour la réparation du système initial. En pratique, nous verrons que cette méthode a des similarités avec un algorithme que nous proposerons dans la seconde partie de cette étude ; dans les cas où cette méthode serait choisie, nous recommanderons donc l'application de cet algorithme pour obtenir des résultats similaires.

V.2.2 Évaluation de l'utilisabilité du processus outillé

Qualité du plan de réparation

Le problème étant un problème de conception sûre et optimale, nous devons évaluer la qualité du plan obtenu selon des critères de sécurité et de coût total.

En termes de sécurité, le gain est indéniable puisque les solutions obtenues sont effectivement garanties comme respectant les contraintes et critères fixés ; cependant, la limitation en contraintes saturées signifie que la solution ne considère aucune marge de manœuvre sur les situations de NO-GO : de telles situations doivent être évitées à tout prix, alors que dans certains cas ces situations sont extrêmement improbables et ne justifient pas de restreindre la solution.

Sur le plan du critère de coût, l'algorithme garantit que sa réponse corresponde au coût optimal

dans la mesure où tous les systèmes sont considérés. Il est possible d'évaluer précisément la valeur ajoutée financière de la méthode dès lors qu'on compare les décisions prises par un opérateur de MCC actuel avec les décisions qui auraient été prises par le processus outillé. Nous n'avions accès lors de cette étude ni à un opérateur de MCC ni à des données opérationnelles ; nous pouvons cependant proposer le protocole d'évaluation de la qualité suivant :

Protocole d'évaluation de la qualité

1. On dispose d'un ensemble de plans de vol ainsi que des défaillances survenues, des décisions prises à chaque instant et du coût total.
2. Ces scénarios sont simulés dans un outil dédié, chaque décision de réparation est résolue au travers du processus outillé. Cette simulation est effectuée automatiquement, sans intervention d'un opérateur. Parmi ces solutions, on interdit les options de déviation.
3. On obtient ainsi une comparaison des coûts pour chaque plan de vol, avec un coût réel et un coût calculé ; la valeur ajoutée est la moyenne de l'écart entre les coûts pour chaque instance.
4. Les instances pour lesquelles la mission n'est pas achevée sont comptées séparément ; on obtient ainsi une proportion de missions achevées dans un cas ou dans l'autre.

Utilisabilité de la solution obtenue

L'autre limitation principale du solveur en termes d'utilisabilité concerne la traduction inverse : un solveur de PCMDP retourne un plan conditionnel, qui est en réalité un processus Markovien. Un plan solution est alors un arbre avec potentiellement autant de branchements qu'il n'y a de pannes possibles. Ceci est inacceptable pour un outil d'aide à la décision, qui doit présenter de manière synthétique les avantages et inconvénients de plusieurs plans de réparation possibles.

Il est donc nécessaire de disposer d'une méthode permettant de résumer automatiquement le plan solution, permettant au MCC, au pilote et à l'opérateur de maintenance de comprendre de manière immédiate et claire quelles sont les options de réparations possibles, quels sont les évènements qui peuvent venir perturber ces réparations, et quels sont les paramètres qu'ils doivent prendre en compte dans leur prise de décision.

Même s'il existe des travaux [DS10] visant à résumer de manière automatique un processus Markovien, par exemple en le factorisant autour de points de passages obligatoires (landmarks) ou en traduisant les valeurs possibles de variables individuelles, nous cherchons à résumer nos solutions sous une forme plus spécifique, s'appuyant sur la connaissance de notre domaine : en se concentrant sur la chaîne d'exécution la plus probable, nous pouvons isoler les actions et schémas importants pour l'opérateur, comme par exemple le fait d'emporter une pièce depuis Berlin pour effectuer une réparation à Nice.

Cependant, notre retour d'expérience sur l'utilisation du démonstrateur a mis en avant l'intérêt d'un dialogue entre le solveur et l'utilisateur : l'utilisateur s'attend en effet à pouvoir poser des questions particulières sur la solution obtenue, par exemple en jouant des scénarios de défaillance hypothétique, ou en demandant de modifier certaines décisions. Plutôt que de *résumer* la solution, une quantité plus importante d'information peut être transmise en permettant *d'interroger* le plan obtenu. Dans le processus outillé global, nous proposons ainsi de réaliser la dernière étape de la manière suivante :

Concept de transmission de la politique solution à l'utilisateur

1. La solution obtenue par le solveur est récupérée par l'outil.
2. Le chemin d'exécution le plus probable est mis en avant en indiquant sur le plan de vol et la ligne temporelle les étapes principales prévues.
3. La présence de plan de contingence est indiqué à l'utilisateur, qui peut librement interroger le système pour évaluer des scénarios de défaillance.
4. Pour chacun des plans et pour la ligne principale, le coût de réparation et la durée totale sont indiqués.
5. Lors d'une décision stochastique, la solution avec le poids le plus fort est privilégié, avec une indication du coût financier associé au choix de chacune des solutions possibles.
6. L'utilisateur peut librement imposer des décisions de réparation, et demander au système une comparaison de cette solution avec le plan optimal.

V.2.3 Faisabilité du processus outillé

En termes d'équipement nécessaire, les tests que nous avons réalisés se sont basés sur un ordinateur de puissance moyenne, tel qu'on peut en trouver dans un environnement de bureau classique. Il est évident qu'un ordinateur plus puissant apporterait des performances meilleures, mais le gain serait vraisemblablement négligeable par rapport à l'augmentation de complexité liée au nombre de systèmes ; un ordinateur plus puissant nous permettrait donc de traiter plus rapidement un exemple particulier, mais ne nous permettrait pas de traiter cet exemple avec un ou plusieurs systèmes supplémentaires.

Ce résultat est aussi valable pour des clusters de calcul : l'algorithme de résolution pour SPC MDP n'est pas optimisé pour le calcul parallèle ; il n'y a donc qu'un avantage très marginal à effectuer du calcul distribué. De plus, la complexité tient plus du nombre d'états explorés que du nombre de boucles réalisées dans les deux parties de l'algorithme (atteignabilité et Value Iteration) ; par conséquent, il semble probable que même une version modifiée de l'algorithme SPC VI pour le calcul parallèle n'apporte que des gains modérés.

Le facteur limitant principal pour les ordinateurs de bureau classiques est en revanche celui de la mémoire : puisque SPC VI réalise une exploration exhaustive de tous les états, devant tous les maintenir en mémoire, il est nécessaire de disposer d'un espace mémoire suffisant. Ainsi, bien que le temps de résolution devrait permettre raisonnablement de traiter des scénarios avec 9 systèmes sur un ordinateur de bureau, nous avons été limité lors de nos tests par la mémoire nécessaire pour traiter plus de 7 systèmes, dépassant les 6 Go disponibles. Nous recommandons ainsi d'équiper un système réel avec le maximum de RAM disponible, par exemple une capacité de 64Go obtenu avec un équipement 4 barrettes de 16 Go de mémoire ; ceci nécessite une carte-mère particulière, mais qui est néanmoins courante dans le commerce.

L'ensemble du processus outillé peut ainsi être effectué sur un même ordinateur, bien que nous recommandons la mise en place d'un système distribué pour la gestion d'une flotte d'avions. Ceci est particulièrement adapté lorsque la majorité des données d'entrée est obtenue par connexion à des bases de données externes.

Enfin, une question critique que nous n'avons pas abordée dans les paragraphes précédents est celui de l'obtention de données précises, permettant de paramétrer les modèles : en effet, en pratique certaines des données requises sont difficiles d'accès, telles que les coûts des différentes actions et leur impact financier. Le calcul de ces modèles de coût et d'impact opérationnel d'une défaillance ou d'un retard au décollage est un domaine à part entière [CTA04], et peut s'avérer être l'un des points majeurs bloquant l'utilisation de notre outil d'aide à la décision.

En particulier, la question se pose de mesurer l'impact d'une imprécision ou de l'absence de données : dans le cadre d'un engagement contractuel envers une compagnie aérienne, il est impératif d'évaluer dans quelle mesure nous pouvons assurer que la solution choisie est optimale, et quel sera l'impact d'une donnée d'entrée erronée sur la solution proposée.

Nous avons déjà évoqué précédemment la manière dont des erreurs de données d'entrée pouvaient être propagées dans la fonction de préférence puis dans la solution optimale, ainsi que trois méthodes permettant de mitiger cette erreur : (1) ne considérer que les paramètres précis, (2) évaluer l'impact de la marge d'erreur sur la solution et (3) proposer plusieurs solutions prenant en compte des marges d'erreur. Néanmoins nous pouvons aussi réfléchir aux protocoles qu'il est possible de mettre en place pour limiter les erreurs lors de l'obtention de données.

Parmi les données d'entrée, nous pouvons isoler trois catégories :

Types de données d'entrée sous l'angle de l'incertitude

- **Les données d'ingénierie** obtenues à partir de documents techniques tels que la MEL et le TSM. La seule incertitude qu'il peut y avoir dans ces données tiendrait du fait que ces documents sont erronés ou ne représentent pas correctement le système ; cette incertitude est donc facilement maitrisable, puisque les corrections doivent être appliquées dès la détection de l'erreur sur les documents.
- **Les données de tierce-partie** obtenues par connexion à des bases de données telles que les plans de vols ou les stocks d'équipement disponibles. La fiabilité de ces données est assurée par le processus d'acquisition des données : une erreur dans les stocks disponibles résulte par exemple d'une erreur dans la saisie des stocks, ce qui peut être corrigé par un processus de gestion des stocks appuyé par des outils informatisés. Nous ne considérerons donc pas ce type d'incertitude.
- **Les données estimées** obtenues à partir d'une expertise, par exemple les valeurs de préférence, les coûts de réparation et les coûts de pénalité de déviation ou de retard. Le principal outil de maîtrise de l'incertitude est ici l'apprentissage par retour d'expérience, qui permet d'évaluer une marge d'erreur sur la valeur estimée. Cependant, ces données présentent une incertitude de façon inhérente, puisqu'ils varient dans le temps : le coût réel d'un retard ne sera pas le même d'une journée sur l'autre, puisqu'il dépendra d'un ensemble de paramètres qui ne sont pas capturés tels que le nombre de passagers, les autres avions en cours d'escale voire les conditions météorologiques.

La prise en compte des données estimées est un des arguments principaux en défaveur de l'utilisation de SPC MDP dans le processus outillé : l'approche la plus fiable pour maîtriser les erreurs d'évaluation des coûts est la réalisation de nombreuses résolutions, prenant en compte les différentes marges d'erreurs, pour informer l'utilisateur de façon exhaustive sur les différentes options possibles ; cependant, le temps de résolution est trop élevé pour permettre cette approche, dans la mesure ou beaucoup de paramètres présentent une telle incertitude. En l'absence de garanties sur l'évaluation des coûts, obtenues par exemple par acquisition de retours d'expérience sur une longue période, nous ne pouvons donc assurer que la solution optimale obtenue par un processus outillé basé sur SPC MDP offre des garanties sur l'optimalité du coût, et donc qu'il soit réellement acceptable dans un contexte industriel.

Notons que des travaux existants se sont intéressés à ce problème, à la fois selon le point de vue de la compensation des erreurs dans la fonction de récompense d'un MDP [DM10] [NHR99], dans la prise en compte des incertitudes sur les transitions [SLJ73] ou plus généralement selon une notion de borne autours des paramètres d'un MDP [GLD00]. Ces travaux n'ont cependant pas été adapté au contexte des systèmes devant respecter des contraintes de sécurité.

V.2.4 Conclusion sur l'évaluation du processus outillé sur le scénario du Business Jet

Dans les chapitres précédents, nous avions identifié et formalisé un scénario d'aide à la décision pour la maintenance avionique sous la forme d'un problème de conception sûre et optimale ; nous avions montré qu'il pouvait être abordé sous différents angles mathématiques et que l'approche la plus complète - celle des processus décisionnels markoviens - n'était pas envisageable sans une méthode de résolution plus performante que les méthodes existantes dans la littérature. Nous avions alors défini un formalisme adapté, appelé SPC MDP, pour lequel nous avions pu développer un algorithme de résolution plus efficace que les algorithmes existants de plusieurs ordres de grandeur.

Dans ce chapitre, l'enjeu était alors de construire un processus outillé complet autour du formalisme SPC MDP, puis de l'appliquer au scénario du Business Jet défini précédemment. Nous avons plus particulièrement réalisé les apports suivants :

- Nous avons montré que la **génération automatique de modèle** permettait de pallier la complexité de modélisation d'un problème SPC MDP.
- Nous avons détaillé des techniques de **capture des documents d'ingénierie et de préférences utilisateurs**, puis mis en œuvre cette capture au travers d'un démonstrateur adapté.
- Nous avons **évalué le modèle SPC MDP** face aux cas du Business Jet ; nous avons montré que notre processus outillé permet de résoudre ce modèle, c'est-à-dire d'obtenir automatiquement un plan de réparation optimal et valide.

Néanmoins, l'évaluation du processus outillé, en particulier sur le plan de la faisabilité, a mis en avant les **difficultés d'obtention de données précises** pour ce cas d'application, en particulier concernant les coûts de chaque action. Ceci implique que l'application industrielle de ce processus s'avère particulièrement complexe, voire coûteuse, si l'on base le processus outillé sur une méthode de type SPC MDP. La conclusion en est que des méthodes d'aide à la décision multi-critères semblent vraisemblablement plus adaptées au contexte économique actuel, bien que des méthodes plus poussées soient plus performantes en termes de réduction des coûts et maîtrise des risques.

Cette conclusion est fondamentale, en ce qu'elle remet en cause la viabilité de notre projet : si tous les problèmes de conception sûre et optimale partagent cette problématique, c'est-à-dire qu'ils nécessitent tous un investissement important pour l'obtention des données face à un gain marginal par rapport à d'autres méthodes, alors ils ne sont vraisemblablement pas adaptés au contexte économique actuel, ce qui amènerait notre étude à conclure que d'autres types de processus sont plus adaptés, tels que les processus basés sur des méthodes "décider puis vérifier".

Il nous faut par conséquent vérifier ou réfuter cette caractéristique sur d'autres cas d'application. Pour cela, nous étudierons dans la seconde partie de cette étude un ensemble de problématiques pouvant être traitées sous l'angle d'un problème de conception sûre et optimale, en évaluant dans quelle mesure l'obtention de données précises nécessite un investissement important et peu rentable pour la mise en œuvre industrielle du processus outillé.

Deuxième partie

Étude du domaine de la gestion de la qualité de service de systèmes critiques

CHAPITRE VI

ÉLARGISSEMENT À UN MODÈLE GÉNÉRAL DE GESTION DE LA QUALITÉ DE SERVICE

Sommaire

Nous avons établi dans la partie précédente quelques caractéristiques essentielles des processus outillés de conception sûre et optimale : nous avons dans un premier temps établi, à partir de l'étude d'un scénario avionique, que la maintenance avionique présente un problème de conception sûre et optimale sous la forme d'un système d'aide à la décision pour la maintenance avionique ; nous avons en particulier conclu que prendre en compte des évènements futurs probabilistes nécessitait l'utilisation de techniques de type "processus décisionnels markoviens".

Nous avons dans un second temps montré qu'il était possible de concevoir un cadre mathématique adapté à la conception sûre et optimale, en choisissant des hypothèses permettant d'avoir des temps de résolutions acceptables sur des modèles industriels.

Nous avons enfin établi un prototype du processus complet, dont les deux principaux résultats sont les suivants :

– La mise en place d'un outil d'aide à la décision pour la maintenance avionique, reposant sur une formalisation des données, permet de traiter une complexité qu'il n'aurait pas été possible de traiter sans assistance.

– Le principal frein à la mise en œuvre d'un tel outil est l'obtention des données, à la fois parce que certaines des données nécessaires sont complexes à acquérir avec une précision suffisante (temps estimés de réparations, préférences et coûts des opérations, ...) et parce que certaines données présentent une incertitude intrinsèque qui ne peut pas être réduite.

Le second point en particulier s'est avéré bloquant pour la suite de notre étude : pour étudier plus en amont ce cas d'application, il aurait fallu étudier les effets de l'incertitude sur les données d'entrée, ce qui aurait nécessité un jeu de données d'entrée réelles. De telles données ne sont à notre connaissance pas disponibles dans la littérature, voire ne sont pas recueillies à l'heure actuelle, et il aurait été improbable de pouvoir les obtenir dans une période de temps réaliste.

Néanmoins, une seconde manière d'étendre les résultats obtenus est de regarder dans quelle mesure ils sont applicables à d'autres scénarios. À défaut d'un scénario unique, nous souhaitons adresser une classe de problèmes, qui peuvent être abordés sous l'angle de la conception sûre et optimale en ce qu'ils présentent à la fois des critères d'optimalité (avoir la meilleure qualité de service) et des contraintes de sécurité (maintenir à tout prix la qualité de service dans une enveloppe, face à un environnement probabiliste).

Nous prenons comme point de départ de notre réflexion la notion de **qualité de service** pour construire cet élargissement. Le choix de cette notion résulte de plusieurs considérations, dont une partie est détaillée en annexe (Annexe D). En conséquence, la démarche que nous suivrons dans ce chapitre consiste dans un premier temps à définir de façon informelle la notion de qualité de service, puis de formaliser cette notion sur le plan de la syntaxe et de la sémantique, et enfin de sélectionner un modèle mathématique adapté permettant de résoudre des problèmes de décision sûre et optimale exprimés dans ce formalisme.

VI.1 Notion de qualité de service

Nous partons de l'hypothèse suivante, concernant la prise de décision, en particulier dans le contexte du pilotage : toutes les décisions prises par le pilote visent à améliorer un ensemble de paramètres, qui sont caractérisés par le fait qu'ils sont mesurables (le pilote peut se rendre compte que ses actions améliorent ou déprécient chaque paramètre), objectifs (deux pilotes différents verront tous les deux de façon manifeste - en dehors des problèmes de perception - lorsqu'un paramètre augmente), munis d'un ordre total (le paramètre peut être clairement identifié comme étant amélioré ou déprécié) et munis d'une borne supérieure dans le sens de l'amélioration (il existe une valeur *idéale* de chaque paramètre). Notons que ceci ne présage en rien de la manière dont le pilote prendra la décision : il peut privilégier l'amélioration de certains paramètres par rapport à d'autres selon une logique qui lui est propre.

Les notions **d'amélioration et de dépréciation** sont primordiales : si nous voulions par exemple choisir l'altitude comme paramètre, il s'agirait d'un mauvais paramètre puisqu'il ne porte pas de notion *d'amélioration*. Un pilote peut ainsi se rendre compte que l'altitude augmente ou diminue, mais il doit faire une traduction interne pour savoir si l'augmentation de l'altitude est une bonne ou une mauvaise chose.

VI.1.1 Définition informelle

Pour étudier plus clairement ce concept de paramètre, nous définissons la notion de qualité de service, de façon très générale telle que dans la définition suivante :

Définition 30 (*Service et Qualité de Service*)

*Un **service** est une fonctionnalité mise à disposition par un système dans l'objectif d'effectuer une tâche particulière.*

*Une **qualité de service** est un paramètre associé à un service, représentant la performance de ce service. Ce paramètre est muni d'un ordre total et possède une valeur maximale.*

Nous voyons par cette définition que les notions de service et qualité de service recouvrent énormément de cas d'utilisation : lorsque le pilote souhaite par exemple contourner une zone à cause d'une contrainte météorologique, il est confronté à plusieurs possibilités de contournement ; certaines de ces solutions privilégient la rapidité de la trajectoire, d'autres privilégient la stabilité de l'avion voire la capacité de l'avion à poursuivre la mission. De la même manière, pour chacune des solutions de contournement possible le pilote aura à prendre un ensemble de décisions, telles que le choix de l'altitude, la vitesse, ou le type de guidage, pour lesquels chaque solution possible aura des paramètres différents - sur la rapidité de la trajectoire et la stabilité de l'avion, mais aussi sur d'autres paramètres tels que la précision des données de position, la capacité de résistance à une défaillance ... Pour recouvrir tous ces cas d'utilisation, la notion de qualité de service a ainsi dû être définie de la manière la plus générale possible, en ne gardant que les plus petites caractéristiques communes de tous ces paramètres.

Notons qu'un lecteur intéressé pourra trouver en annexe des considérations supplémentaires, qui étendent la définition informelle de la notion de qualité de service dans le contexte de la prise de décision (Annexe E).

VI.1.2 Étude de concepts fondamentaux des langages de modélisation existants

Pour formaliser les problèmes de décision, nous devons donc à présent formaliser la notion de qualité de service, ce qui implique de formaliser les autres composantes de la définition informelle : puisque la qualité de service est associée à un service, lui-même fourni par un composant, il nous faut à présent définir en termes mathématiques ce qu'est un composant, un service, une qualité de service, ainsi que la manière dont ces éléments sont liés les uns aux autres.

Pour cela, il est possible de s'intéresser aux concepts manipulés par les langages de modélisation existants : nous avons déjà évoqué dans l'état de l'art le langage Altarica [APGR99], qui a lui-même évolué pour former plusieurs versions existantes [Pro14] ; Altarica est construit autour de composants élémentaires, appelés "noeuds", contenant chacun des entrées et des sorties appelées "variables de flux" ; chaque noeud contient de plus une machine à états, constituée de "variables d'états" et de

"transitions" (Annexe F). Nous retrouvons donc ici des concepts que nous avons déjà évoqués : un composant élémentaire (un noeud) fournissant un certain nombre de services (des flux) dont les valeurs représentent une qualité de service que l'on souhaite maintenir correcte vis-à-vis de certains critères. Nous pouvons donc voir que le langage Altarica permet tout à fait de porter un modèle centré sur la qualité de service.

C'est aussi le cas du langage de modélisation sysML [FMS14], qui est tout comme Altarica un des langages de modélisation principal utilisé par la communauté de modélisation formelle : sysML contient parmi ses concepts plusieurs notions de "blocs" fournissant chacun des informations aux travers de "ports" ou encore de "pin" ; cependant, il est important de noter que sysML est conçu comme un ensemble de plusieurs diagrammes, apportant chacun une information précise, tels que les diagrammes d'exigences, les diagrammes de définition de bloc ou encore les diagrammes d'activité. En ce sens, il semble plus complexe d'adapter sysML pour porter un modèle centré sur la qualité de service, puisque sysML n'a pas pour objectif de produire des modèles pouvant être simulés ; ceci est partiellement compensé par le fait que sysML peut être associé à des langages de modélisation permettant la simulation [VD06], tels que Modelica ou Simulink, ou lorsque les outils implémentant le formalisme sysML proposent cette simulation [HRM07].

Dans une certaine mesure, de nombreux langages de programmation peuvent aussi porter un modèle centré sur la qualité de service : la plupart des langages Orientés Objets sont construits autour d'un ensemble d'éléments, dont les flux sortants permettent de définir un ensemble de services fournis, associés à une certaine qualité. Ceci est particulièrement le cas dans des langages de programmation reposant sur des concepts tels que le **design by contract** [Mey92], comme les langages Eiffel ou encore Ada, qui stipulent que chacune des entrées et sorties doit être associée à un ensemble de conditions précisant la nature des données échangées, ce qu'il est possible de faire comme opération sur ces données ainsi que des conditions sur les valeurs que ces données doivent avoir. Néanmoins, ces langages n'ont pas été créés spécifiquement pour modéliser des systèmes sous l'angle de la qualité de service ; nous pouvons donc nous attendre à ce qu'ils soient beaucoup trop complexes pour notre besoin, puisque nous n'utiliserions qu'une faible partie du langage - celle nous permettant de spécifier des contraintes sur les informations.

La démarche que nous avons décidé de suivre, dans la section suivante, a été de définir formellement les concepts nécessaires à une modélisation d'un problème de décision sous l'angle d'une gestion de la qualité de service, sans nous attacher à un langage particulier. Comme nous le pressentons au travers de ce court état de l'art des langages existants, et comme nous le verrons par la suite, le modèle que nous proposons peut être implémenté dans de nombreux formalismes et peut être traduit aisément d'un formalisme à l'autre.

VI.2 Définition mathématique du modèle de gestion de modes dégradés

Dans les paragraphes suivants, nous définissons un modèle que nous appellerons **Modèle de gestion de modes dégradés (GMD)**, dont la caractéristique principale est qu'il permet de représenter l'impact d'une défaillance ou d'une dégradation sur un système, sous la forme d'un impact sur les qualités de service produits.

Pour cela, nous nous basons sur la définition informelle que nous avons établie précédemment, qui présente les concepts de système et de service. Ceci correspond à la vision la plus naturelle lorsqu'on parle d'un objet dans un cadre industriel : nous décrivons simplement une *boîte noire* qui *fait* quelque chose. Si nous voulons rentrer dans le détail, nous pouvons alors dire que cette boîte est composée d'autres boites, chacune d'entre elles faisant une ou plusieurs choses, de façon à ce que l'ensemble des choses faites par les boites de plus bas niveau se traduise en une des choses faites par la boite de plus haut niveau.

VI.2.1 Ressource, Mode, Service

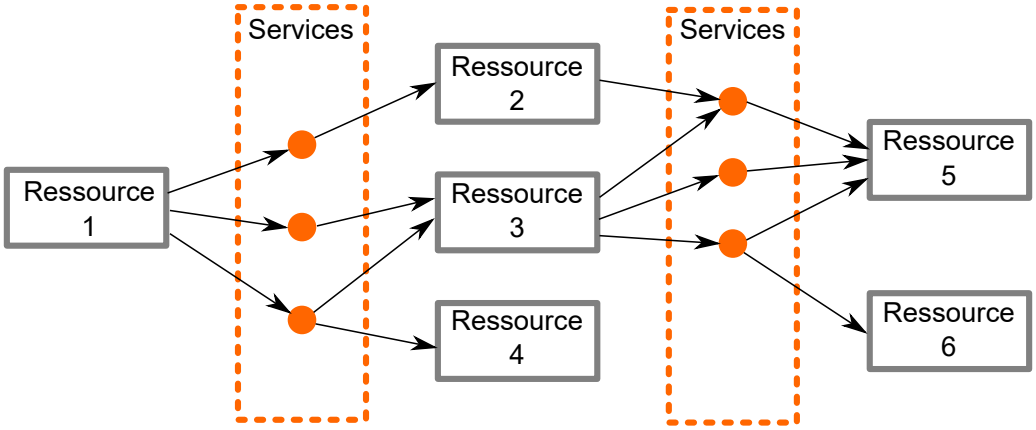


FIGURE 22 – Modèle GMD : plusieurs ressources sont liées par des services fournis ou utilisés.

Pour clarifier le discours, nous appellerons **Ressource** un élément quelconque, élémentaire ou non. Nous avons naturellement envie de dire qu'une ressource peut contenir d'autres ressources, puisque cela correspond à des usages répandus dans les communautés de modélisation, tels que le raffinement qui consiste à obtenir un niveau de détail de plus en plus précis en détaillant le contenu de chaque ressource ; cependant, comme nous le verrons par la suite il n'est pas nécessaire d'imposer une vue hiérarchique directement dans la syntaxe du modèle : comme nous nous intéressons à la propagation des défaillances dans un système, nous n'utilisons pas des informations telles que l'inclusion d'une ressource dans une autre, toutes les ressources sont donc *mises à plat*, aussi bien les ressources "mères" que les ressources "filles". Notons néanmoins que la vision hiérarchique peut tout à fait être portée par un outil implémentant la capture et l'exploration du modèle.

Nous appellerons alors **Services** les flux d'informations entrants et sortants permettant de caractériser ce qui est fait par une ressource. Ces flux ne sont pas en général des flux de données : on parlerait par exemple plutôt de *signal* si on voulait dire qu'une sonde de température a pour sortie un signal avec une valeur de 18 degrés ; à la place, nous nous intéressons uniquement au service, en disant que cette sonde de température fournit un service "mesure de la température", qui peut avoir plusieurs caractéristiques (précis ou imprécis, évaluation instantanée ou différée, voire tout simplement présent ou absent,...) (Figure 22 page 129). Le nom de "service" est cohérent avec des définitions usuelles dans le domaine [ALRL04], qui utilisent ce terme pour désigner le comportement d'un système perçu du point de vue de l'utilisateur.

Enfin, nous appellerons **Modes** d'une ressource les configurations dans lesquelles cette ressource peut se trouver. L'une des nuances essentielles est que le mode représente l'état interne de la ressource, indépendamment de la valeur des services entrants et sortants. Ainsi, si une ressource GPS n'est pas

alimentée, il n'est pas correct de dire qu'elle a changé de mode : son état interne reste inchangé - la ressource GPS n'est pas en panne - même si évidemment elle n'est plus capable de fournir les services attendus. Ainsi, un mode représente un état élémentaire d'une ressource, qui ne peut pas être influencé par une autre ressource extérieure. De même, une ressource a un seul mode, puisqu'elle se trouve à chaque instant dans un seul état interne ; ceci va à l'encontre des choix de design d'autres formalismes tels qu'Altarica, pour lesquels un composant peut avoir plusieurs variables de mode. Dans notre formalisme, si nous souhaitons exprimer qu'une ressource a plusieurs variables de mode, alors c'est qu'il est nécessaire d'inclure un niveau de détail supplémentaire, puisque cela veut dire que la ressource est en fait divisée en plusieurs ressources qui ont chacune un mode interne.

Ces considérations sont résumées dans la définition suivante :

Définition 31 (*Ressource, Mode, Service*)

On définit :

$E_{ressource}$ *l'ensemble des* **ressources**, $i \in E_{ressource}$ *une ressource, c'est-à-dire une partie physique ou fonctionnelle d'un système.*

M_i *est l'ensemble des* **modes** *d'une ressource, c'est-à-dire tous les modes de fonctionnement ou modes dégradés dans lesquels la ressource i peut se trouver ; on note* m_i *le mode courant de i.*

S *est l'ensemble des* **services**, $s \in S$ *est un service qui peut être fourni ou utilisé comme entrée d'une ressource, c'est-à-dire un flux d'informations physique ou virtuel. On note* $S_i^{(in)}$ *et* $S_i^{(out)} \subset S$ *respectivement les services entrants et sortants d'une ressource.*

Q_s *est l'ensemble des* **qualités de service** *possibles pour* s, q_s *étant la qualité de service courante du service* s*, c'est-à-dire une caractérisation de la qualité de l'information dans* s *;* Q_s *repose sur une relation d'ordre total* $<_s$*. On définit la notation* Q_S *comme le produit de tous les* Q_s *pour tout* $s \in S$*.*

VI.2.2 Qualité de service garantie et attendue

Dans le paragraphe précédent, nous avons défini des briques individuelles, les ressources, ainsi que les informations entrantes et sortantes ; il est donc naturel de connecter à présent les différentes ressources, ce qui est fait en connectant les services entrants et les services sortants. Cette connexion pourrait se faire de plusieurs manières, par exemple de la façon traditionnelle en parlant de qualité de service reçue par un système et de qualité de service produit, puis en disant que sur un lien de communication entre les ressources ces deux qualités sont égales. Ceci est le cas dans des langages centrés sur les signaux tels qu'Altarica ou Simulink. Cependant, nous avons choisi une autre approche inspirée par la notion de **contrat** ([Mey92] [SJ13]) : nous nous intéressons à la propagation d'une défaillance dans un système en l'envisageant comme la rupture d'un contrat. Plus précisément, un lien entre deux ressources représente un contrat liant deux services : le fournisseur s'engage à fournir un service avec une certaine qualité et le client attend un service avec une certaine qualité. Ceci est rendu possible par le fait que les qualités de service sont construites autour d'un ordre total. Pour un même lien entre deux ressources (liant deux services), nous avons donc trois qualités de service : celle qu'on espère fournir, celle qui est réellement fournie et celle qu'on s'attend à recevoir ; le contrat est respecté lorsque la qualité fournie réellement est supérieure ou égale à la qualité qu'on s'attend à recevoir (Figure 23 page 131).

Pour formaliser cette notion, nous définissons d'un côté la qualité de service garantie, qui est une qualité de service minimale que s'engage à fournir la ressource (bien qu'elle puisse fournir une qualité plus grande, voire plus faible comme nous le verrons), ainsi qu'une qualité de service attendue de l'autre côté. Ces qualités dépendent uniquement du mode de la ressource : il est par exemple possible d'imaginer des modes "économie d'énergie", où une ressource attend en entrée une qualité de service sur l'énergie qui est plus faible qu'en mode nominal, mais où elle fournit des services (par exemple un service GPS) avec une qualité garantie plus faible.

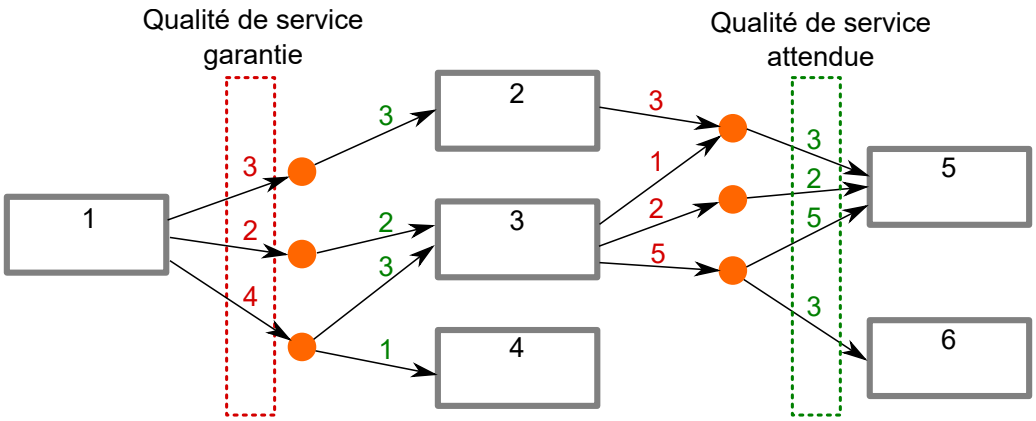


FIGURE 23 – Modèle GMD : la qualité de service est un contrat entre les services sortants et les services entrants.

Définition 32 (*Qualité de service garantie et attendue*)

Soit une ressource $i \in E_{ressource}$, soit un mode de cette ressource $m \in M_i$,
*$m^{(in)} :\rightarrow Q_S$ et $m^{(out)} :\rightarrow Q_S$ sont respectivement les **qualités de service attendue** et les **qualités de service garanties** lorsqu'une ressource i est dans le mode m. Une ressource reçoit des services en entrée, chacun avec une qualité attendue, et produit des services en sortie avec une qualité garantie.*

On note $q_{s|i}$ la qualité réelle du service s directement fournie par i.

Nous pouvons donc voir qu'une première étape de la construction d'un système est de faire correspondre les contrats : pour chacun des liens entre les services, il faut que les qualités de service garanties soient plus grandes que les qualités de service attendues ; ceci peut par exemple être effectué en sélectionnant les modes appropriés pour chaque ressource - ou en changeant l'architecture.

Cependant, la qualité de service réelle des services peut varier : elle dépend à la fois du mode courant de la ressource et de la qualité des services entrants. En vérité, au regard des définitions précédentes, il n'est pas nécessaire de connaître précisément la valeur de la qualité de service réelle, puisqu'il suffit de savoir si elle respecte ou non la qualité de service garantie. Pour un mode fixé, la qualité de service réelle n'a donc que deux valeurs possibles : soit elle est égale à la qualité de service garantie *lorsque tout va bien*, soit elle est égale à la valeur minimale ; on ne cherche donc pas à calculer à quel point elle est plus grande que la qualité de service garantie lorsque tout va bien, ni à calculer à quel point elle est en dessous de la qualité garantie lorsque quelque chose n'est pas correct.

Lorsqu'au moins l'une des qualités de service réelles q_s d'un service s est strictement inférieure à l'une des qualités de service attendues $q_s <_s m_i^{(in)}(s)$, on dit alors que la ressource i est **inconsistante** ; il y a rupture du contrat entre les deux services. La ressource i ne peut alors fournir aucun de ses services produits avec la qualité garantie (Figure 24 page 132). En conséquence, par convention, la qualité réelle de chacun des services produits par la ressource i est fixée à la valeur minimale de la qualité de service :

Définition 33 (*Première règle de mise à jour : inconsistance*)

$$\forall i \in E_{ressource}, \left(\exists s \in S_i^{(in)}, q_s <_s m_i^{(in)}(s)\right) \Rightarrow \left(\forall s' \in S_i^{(out)}, q_{s'|i} = \min(Q_{s'})\right)$$

Par opposition, si tous les services entrants ont la qualité attendue, une ressource i fournira tous ses services avec la qualité garantie.

Définition 34 (*Seconde règle de mise à jour : consistance*)

$$\forall i \in E_{ressource}, \left(\forall s \in S_i^{(in)}, q_s \geq_s m_i^{(in)}(s)\right) \Rightarrow \left(\forall s' \in S_i^{(out)}, q_{s'|i} = m_i^{(out)}(s')\right)$$

Pour évaluer quelle qualité de service une ressource reçoit pour un service donné, nous souhaitons intuitivement dire qu'elle sélectionne la meilleure source disponible pour ce service : pour un service donné, la qualité réelle est obtenue à partir du maximum de toutes les qualités produites, en considérant

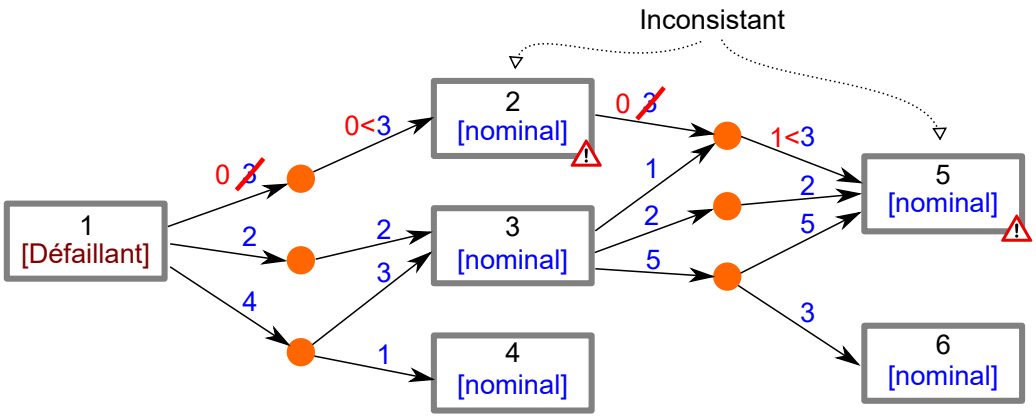


FIGURE 24 – Modèle GMD : suite à un changement de mode, certains contrats peuvent être rompus, ce qui change plusieurs qualités de service mais ne change pas les modes des autres ressources.

toutes les ressources fournissant ce service. Ainsi, lorsque plusieurs ressources produisent un même service, ceci suppose que la première opération réalisée dans chaque ressource recevant ce service est de sélectionner la source appropriée.

Définition 35 (*Troisième règle de mise à jour : consolidation*)

$\forall i \in E_{ressource}, \forall s \in S_i^{(in)}, q_s = \max_{i \in E_{ressource}}(q_{s|i})$

Si aucune ressource ne fournit le service s, la qualité de service de s est fixée au minimum de la qualité de service.

On peut prouver facilement que ces règles de mise à jour convergent lorsqu'elles sont itérées sur un système : en construisant un vecteur contenant toutes les qualités de service réelles, ce vecteur est décroissant à chaque mise à jour (en fonction de l'ordre total déterminé par la combinaison des ordres totaux sur les qualités de service). Associé au fait que si le vecteur après mise à jour est identique au vecteur avant mise à jour, alors il restera identique après l'application de chacune des règles de mise à jour, ceci prouve la convergence de la mise à jour.

Cependant, l'état final n'est pas unique : il dépend à la fois du système et de l'état initial, c'est-à-dire de la valeur des qualités de service avant le début de l'algorithme de mise à jour. Pour résoudre ce problème, nous définissons le processus de mise à jour suivant (les modes m_i de chaque ressource i étant fixés) :

Algorithm 11: Mise à jour de la qualité de service

```
1 Fixer toutes les qualités de service réelles q_s à leur valeur maximale max(Q_s) ;
2 while l'un des q_s a changé do
3     Appliquer les règles de mise à jour 1 à 3 ;
```

L'unicité de la limite peut alors être prouvée, puisque l'état final ne dépend pas de l'ordre dans lequel les ressources sont mises à jour, ni de l'ordre dans lequel les règles sont appliquées.

Théorème 8 (*Unicité de la limite de convergence*)

L'algorithme de mise à jour (Algorithme 11 page 132) converge vers une valeur unique pour toutes les qualités de service, pour un ensemble de modes fixé pour toutes les ressources ; cette limite est indépendante de l'ordre d'application des règles et de l'ordre de mise à jour des ressources dans l'algorithme.

Démonstration : En effet, si une séquence finie d'application de règle *Seq* mène à une limite donnée L (c'est-à-dire un ensemble de qualités de service, tel que l'application de toute nouvelle règle ne change aucune valeur), alors on peut montrer que l'ajout de toute nouvelle application de règle à n'importe quel endroit de *Seq* conserve toujours la même limite : par récurrence, soit *Seq* une séquence finie composé de $\sigma_1, \ldots, \sigma_i, \ldots, \sigma_n$ où chaque σ est l'application d'une règle 1,2 ou 3 sur une seule ressource ; l'ajout

de toute opération σ juste après σ_i est supposé par hypothèse ne pas changer la limite ; alors puisque σ_i est l'application de la règle 1, 2 ou 3 sur une ressource ou un service, σ_i et σ sont commutatifs de façon triviale dans les cas suivants :

- σ_i est une règle 1 (inconsistance) ou 2 (consistance) et σ est une règle 1 ou 2 ou n'impacte aucun des services entrants de la ressource concernée, et réciproquement σ_i n'impacte aucun des services sortants de la ressource concernée par σ.
- ou inversement σ_i est une règle 3 (consolidation) et σ est aussi une règle 3, ou n'impacte aucun des services sortants de la ressource concernée, et réciproquement.

Il reste donc deux cas (avec les conditions réciproques pour σ) à étudier : si σ_i est une règle 1 ou 2 sur une ressource i et que σ est une règle 3 sur une ressource j ayant un effet sur au moins l'une des qualités de service entrants de la ressource concernée, alors c'est que σ_i dégrade l'une de ces qualités de service, ce qui peut potentiellement déclencher les conditions nécessaires aux règles 1 ou 2, dégradant une ou plusieurs qualités de service sortants de i. Notons Q l'ensemble de ces qualités de service. Or, par hypothèse l'ajout de $\sigma\&\sigma_i$ juste après σ_i ne change pas la limite. En appelant Q' les qualités de service de la limite, nous pouvons donc établir que toute qualité de service de Q est supérieure ou égale à la valeur de la qualité de service correspondante dans Q' : l'application de σ avant σ_i n'a donc pas dégradé les qualités de service plus bas que ce qu'elles auraient été baissées sans cet ajout, ce qui montre donc que la commutation de σ et σ_i n'a eu aucun effet sur la limite.

Avec un raisonnement similaire, nous pouvons montrer la récurrence sur le cas où σ_i est une règle 3 et que σ est une règle 1 ou 2. Comme la condition initiale de récurrence est vraie puisque la limite est inchangée par toute nouvelle opération, ceci montre la récurrence dans le cas général.

Alors, si Seq_1 est une séquence menant à une limite L_1 et Seq_2 est une séquence menant à une limite L_2, la séquence $Seq_1\&Seq_2$ mène à la limite L_1 puisque l'ajout de Seq_2 après Seq_1 ne change pas la limite, mais mène aussi à la limite L_2, puisque l'ajout de toute opération à la séquence Seq_2 ne modifie pas la limite. Ceci montre que $L_1 = L_2$, ce qui prouve l'unicité de la limite de convergence de l'algorithme. ■

Il est important de noter que ce résultat vient du fait que notre définition des relations entre les services entrants et les services sortants est particulièrement restrictive : dans la plupart des langages de modélisation, les données entrantes et sortantes sont liées par des équations plus complexes, permettant une plus grande expressivité mais ayant le défaut supplémentaire de permettre des états indéterminés. C'est par exemple le cas d'un composant dont la sortie *out* serait $out = (1 - in)$, où *in* serait une entrée connectée directement à la sortie ; un algorithme de mise à jour ne s'arrêterait donc jamais (si *out* et *in* ne peuvent prendre pour valeur que 0 ou 1), puisqu'il assignerait successivement des valeurs 0 puis 1 à l'entrée et la sortie, un peu comme un composant qui clignoterait en permanence.

Notre modèle n'est pas confronté à ce problème, mais avec la limitation ajoutée qu'il n'est pas possible d'exprimer certaines relations entre les entrées et sorties du système ; cependant, la gestion de la qualité de service est un problème sensiblement différent de la gestion des signaux circulant dans un système : il est sensé dans le contexte de la qualité de service d'imposer que la diminution d'une donnée d'entrée n'implique jamais l'augmentation d'une donnée de sortie. Dans le cas contraire, ceci signifierait en effet que la baisse de qualité d'un service fourni a un effet bénéfique sur d'autres qualités de service, ce qui n'est vraisemblablement pas le cas.

VI.2.3 Transitions exogènes et contrôlables

Enfin, nous devons définir dans quelles conditions chaque ressource est autorisée à changer de mode : nous avons déjà évoqué précédemment le fait que le mode correspondait à un état interne de la ressource ; la liste des modes est exhaustive, ce qui signifie que la ressource ne peut pas se trouver dans un autre état que ceux définis par les modes.

Nous retrouvons ici une notion de machine à états, similaire à ce qu'il est possible de trouver dans des langages tels qu'Altarica : la ressource contient plusieurs états, et il est possible de passer de l'un à l'autre instantanément au travers de transitions.

Définition 36 (*Transition*)

On définit une ***transition*** *comme étant un changement de mode d'une ressource. Par définition, une transition ne concerne qu'une seule ressource, puisqu'elle correspond à un effet unique et local sur une ressource donnée, tel qu'une panne ou une reconfiguration.*

Soit T_i l'ensemble des transitions possibles d'une ressource i ; t_i est une transition définie par :

- *Sa garde $t_i^g : M_i \times S_i^{(in)} \times S_i^{(out)} \rightarrow \{0,1\}$, qui définit l'ensemble des modes et valeurs de service possibles dans lesquels cette transition peut se produire.*
- *Son effet $t_i^e \in M_i$, qui définit le mode d'arrivée suite à la transition.*

Contrairement à certains langages qui explicitent la cause d'une transition, par exemple au travers de la notion "d'évènement" dans le langage Altarica, nous faisons ici de la transition un objet élémentaire : une transition n'est pas provoquée par quelque chose, mais est à l'inverse l'évènement racine provoquant des changements de qualité de service en cascade. Puisque la transition est un objet élémentaire, il est indispensable de lui associer plusieurs caractéristiques qui permettent de décrire sous quelles conditions cette transition apparaît par elle-même. Nous faisons ici la différence entre les transitions qui ont une certaine probabilité de se déclencher spontanément et les transitions qui sont déclenchées à dessein par l'utilisateur :

On définit une **transition exogène** comme étant une transition survenant sans contrôle de la part du système, par exemple une panne ou un changement d'environnement. Une telle transition t correspond à un changement forcé de mode dans une ressource, et est associée à un **taux de défaillance** noté λ_t. Ces transitions sont donc supposées indépendantes du temps. Notons, à titre de remarque, que cette vision est très proche du type de considérations qu'il est possible d'avoir dans le domaine des systèmes embarqués, mais qu'elle peut être trop restrictive dans d'autre domaines, par exemple des domaines nécessitant des transitions périodiques ; pour de tels domaines, une étude spécifique sera donc nécessaire pour adapter les preuves suivantes à ces nouvelles caractéristiques.

On définit une **transition contrôlable** comme une transition pouvant survenir dans une situation donnée, mais qui est initiée par le système en fonction de la situation correspondante. Une telle transition représente une reconfiguration fonctionnelle ou opérationnelle.

VI.2.4 Traduction vers un processus décisionnel markovien à temps continu

La définition des transitions nous permet de préciser la dynamique du système : nous avons évoqué dans les paragraphes précédents le fait que certaines transitions étaient associées à un taux de défaillance, représentant une probabilité que la transition arrive par elle-même. Ce concept est en réalité inspiré des processus décisionnels markoviens, dont les processus décisionnels markoviens à temps continu sont l'une des formes les plus générales.

On rappelle la définition des processus décisionnels markoviens à temps continu (CTMDP [1]) :

Définition 37 (*Processus décisionnel markovien à temps continu (rappel)*)

Un processus décisionnel markovien à temps continu et à espaces d'actions et d'états finis est un vecteur (S,A,R,q,I) où :

S est un espace d'états.

A est un ensemble dénombrable d'actions.

$R : S \times A \rightarrow \mathbb{R}$ est une fonction de récompense.

$q(s'|s,a) : S \times A \times S \rightarrow [0;1]$ est une fonction de transition, donnant un taux de transition vers l'état s' lorsque le système se trouve dans l'état s et applique l'action a.

$I : S \rightarrow [0;1]$ est une distribution de probabilité initiale.

Notons que puisque nous imposons un nombre fini de valeurs pour les modes des ressources, les espaces d'états et d'actions sont finis. On définit alors la sémantique principale du modèle au travers de la traduction suivante :

1. Continuous Time Markov Decision Process

Définition 38 (*Traduction en MDP à temps continu*)

On définit le processus décisionnel markovien à temps continu (S, A, r, q, I) *associé à un modèle de gestion de modes dégradés de la manière suivante :*

- S *l'espace des états est défini par* $M_0 \times \ldots \times M_N$ *l'ensemble de tous les modes composés possibles.*
- A *l'espace des actions est l'ensemble de toutes les transitions contrôlables.*
- R *la récompense est une fonction de préférence sur les valeurs des services dans l'état où le système se trouve immédiatement après avoir appliqué l'action* a *dans l'état* s*, multipliée par l'espérance du temps moyen passé dans cet état avant une transition exogène.*
- q *la fonction de transition est définie telle que pour* $q(s, a, s')$*, si on note* $s_{|a}$ *le résultat de l'application de l'action* a *dans l'état* s*, alors* s' *est le résultat de l'application d'au plus une seule transition exogène aléatoire à partir de* $s_{|a}$*.* q *est ici la fonction de taux de transition, et a par conséquent exactement la valeur* $\sum_{e \in D} \lambda_e$ *où* D *est l'ensemble de toutes les transitions exogènes pouvant survenir dans* $s_{|a}$ *et menant à* s'*.*
- I *est un état initial spécifié.*

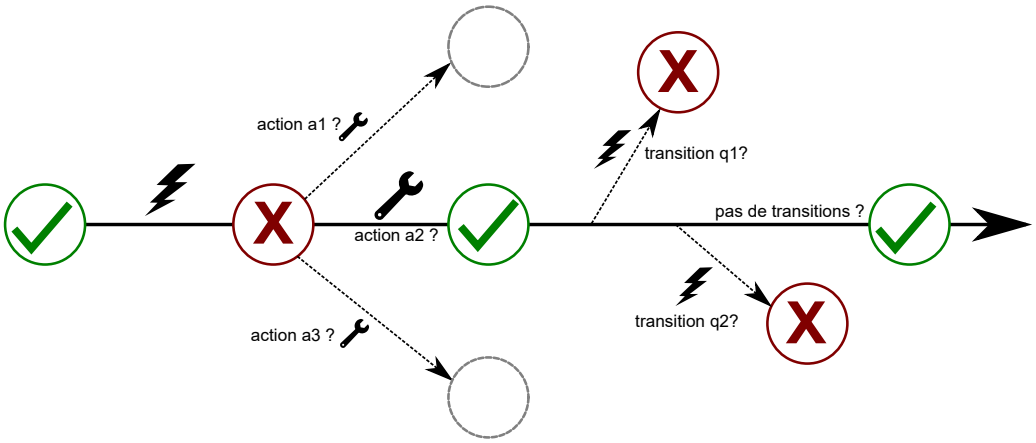


FIGURE 25 – Modèle GMD : une défaillance d'une ressource nous amène dans un état incorrect ; une action a_2 peut être prise pour récupérer une partie des capacités, puis on envisage tous les cas de figure : certaines pannes surviennent avant qu'on ait pu effectuer une autre action, ou rien ne se passe et l'agent est libre d'effectuer une action supplémentaire s'il le souhaite.

Trouver une politique optimale revient ainsi à trouver une politique permettant au système de fournir aussi longtemps que possible, et dans une qualité aussi bonne que possible, les services sélectionnés par la fonction de préférence. La solution de ce CTMDP (Figure 25 page 135) est une politique de contrôle π où $\pi(s)$ est l'action que le système doit prendre dans l'état s.

Il est cependant important de noter que le choix d'une fonction de récompense numérique n'est pas nécessairement le choix le plus pertinent : il est en vérité plus adapté de parler de modèles de préférence, à la place d'une valeur de récompense comme critère de planification [JM+09], puisqu'il n'est pas toujours sensé de traduire une qualité de service en valeur numérique. Par exemple, si une qualité de service décrit une précision entre 0 et 1, tandis qu'une autre qualité de service décrit un temps de réponse (aussi entre 0 et 1), il n'est à priori pas correct d'effectuer des opérations sur ces deux qualités de service, comme en faire la somme pondérée ou la somme cumulative dans le temps.

Néanmoins, dans la suite de cette étude nous faisons l'hypothèse qu'il existe un critère numérique sur les qualités de service instantanées ; cette hypothèse est réalisée pour deux raisons : (1) dans les cas industriels que nous avons rencontrés, les exigences de l'utilisateur sur les qualités de service étaient suffisamment simples pour pouvoir être exprimées sous la forme d'un critère numérique, et (2) nous présentons ici un modèle extrêmement simple, qui a pour vocation de mettre en avant certains concepts innovants, tout en permettant d'être étendu par d'autres techniques existantes dans l'état de l'art, telles que l'observabilité partielle, des modèles de contraintes ou de récompense différentes, ou des variables d'état continues.

On prend l'hypothèse que la décision se fait instantanément et qu'aucune transition exogène ne survient avant la fin d'une action en cours, c'est-à-dire avant que la transition contrôlable qui a été choisie par le système ne s'achève. Cette hypothèse est potentiellement limitante en ce qu'elle ne permet pas d'exprimer qu'une action puisse échouer ou être interrompue par un changement de contexte. Cette hypothèse nous semble cependant raisonnable pour deux raisons :

- Dans une très grande majorité des cas, la durée d'une action est à une échelle de temps sensiblement plus faible que la durée moyenne entre deux défaillances : l'interruption d'une action est alors une transition dont la probabilité de se produire est bien plus faible que toutes les contraintes de sécurité, ce qui implique qu'il est possible d'ignorer la contribution de ces futurs possibles à la validation ou l'invalidation des contraintes.
- Une action qui échouerait serait une action qui aboutirait dans un état autre que l'état attendu ; ce raisonnement masque la raison pour laquelle une action pourrait échouer, par exemple une erreur dans le code informatique qui donnerait dans certaines situations un résultat non attendu. Cette erreur ne résulte cependant pas d'un changement de mode imprévu, mais d'une information qui n'était pas conforme à ce que l'on s'attendait avant d'effectuer l'action - par exemple un service d'entrée qui n'est pas dans l'intervalle attendu, potentiellement parce qu'en amont une transition exogène n'a pas été perçue, provoquant un comportement non attendu dans le code. Ainsi, parler de l'échec d'une action masque en réalité un changement de valeur de service qui n'était pas perçue par le système : ceci montre que cette hypothèse est un des cas où restreindre les capacités du langage apporte un avantage réel pour la qualité du système, en ce qu'elle met en valeur une ambiguïté qui n'était pas perçue auparavant - bien qu'elle représente de fait plus de travail lors de la modélisation.

Définition 39 (*Hypothèse de non interruption*)

Aucune transition exogène ne survient entre la fin d'une transition exogène et la fin de l'action choisie en réponse par le système. Ceci comprend notamment le temps pris par le système pour décider de l'action ainsi que la transition contrôlable composant l'action. On considèrera donc qu'une action est choisie et appliquée immédiatement après l'application de chaque transition exogène - cette action pouvant être l'action vide de transitions.

VI.2.5 Simplification vers un Processus Décisionnel Markovien à temps discret

De façon classique [Put94], on peut définir le processus de saut associé à ce CTMDP : l'objectif est de simplifier le processus en ne s'intéressant qu'aux périodes de décision ; pour cela, nous construisons un processus décisionnel markovien à temps discret, pour lequel chaque étape correspond à un instant de décision suivant une défaillance. De cette manière, nous condensons en un seul état toute la dynamique du système qui pourrait se produire entre deux instants de décision ; ceci est rendu possible en particulier parce que les évènements suivent des lois exponentielles - il nous faudrait sinon associer un processus de Poisson décrivant les instants de saut pour garder toutes les nuances du modèle.

Dans le MDP à temps discret qui est construit (Discrete-Time MDP), les probabilités des transitions sont modifiées : la probabilité ne représente plus l'espérance sur la probabilité qu'une certaine transition se déclenche par elle-même dans un certain incrément de temps (i.e. le paramètre de la loi de probabilité), mais représente une compétition entre toutes les transitions possibles ; on cherche ainsi à savoir quelle transition se produira en premier, ce qui implique que la nouvelle probabilité de transition $T(s, a, s')$ est un rapport entre les taux $q(s, a, s')$ de cette transition et la somme des autres taux.

Lemme 6 (*Traduction en DTMDP*)

On définit le processus décisionnel markovien suivant :

S *l'espace d'états demeure identique :* $M_0 \times \ldots \times M_N$ *est l'ensemble de tous les modes possibles.*

A *l'espace des actions demeure aussi le même, l'espace des transitions contrôlables.*

$\mathring{R}$ *la fonction de récompense où* $\mathring{R}(s,a) = f(s_{|a})E(t(s_{|a}))$ *telle que définie précédemment, où* $s_{|a}$ *est l'état obtenu immédiatement après l'exécution de l'action* a *dans l'état* s *et* $E(t(s_{|a}))$ *est le temps moyen (i.e. espérance) où le système restera dans l'état* $s_{|a}$*.*

T *la fonction de transition, définie telle que dans* $T(s,a,s')$ *, s' est le résultat de l'application de l'action* a *dans l'état* s*, suivie exactement d'une transition exogène ; T a pour valeur :* $T(s,a,s') = \frac{\sum_{e \in D_{s_{|a}}} \lambda_e}{\sum_{e \in D} \lambda_e}$ *où* D *est l'ensemble de toutes les transitions exogènes pouvant survenir dans* $s_{|a}$*, et* $D_{s_{|a}}$ *est l'ensemble de toutes les transitions exogènes pouvant survenir dans* $s_{|a}$ *et menant à* s'*.*

I *est un état initial spécifié.*

Ceci décrit un processus décisionnel markovien où chaque transition correspond à l'apparition d'une transition exogène, et où chaque action est prise de façon à réagir exactement à la situation courante, c'est-à-dire au fait de mettre le système dans le meilleur état possible.

Cependant, bien qu'elle permette de créer un processus décisionnel markovien qui soit très simple, cette définition porte en elle une certaine incohérence - du fait de la restriction de la fonction de transition à exactement une transition exogène : en effet, cette définition ne prend pas en compte le fait que plusieurs transitions contrôlables non simultanées sont parfois nécessaires pour amener le système dans un état satisfaisant (Figure 26 page 137).

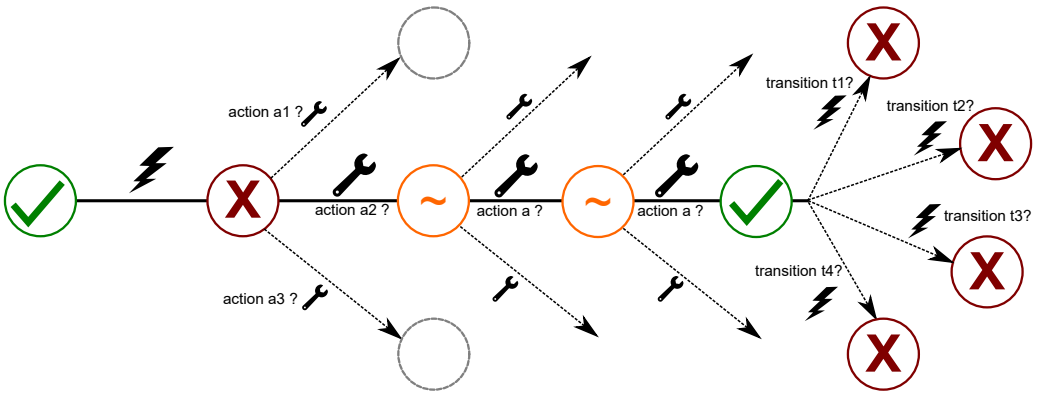


FIGURE 26 – Modèle GMD : une défaillance d'une ressource nous amène dans un état incorrect ; une procédure peut être prise pour amener le système dans un état acceptable, puis aucune action supplémentaire n'est prise jusqu'à ce qu'une nouvelle défaillance ne survienne.

Pour remédier à cela, nous devons modifier l'espace d'actions, pour en faire un espace sur les procédures : une procédure est une succession indivisible d'actions permettant d'amener le système dans un meilleur état, par exemple l'extinction d'un équipement défaillant suivie du démarrage de l'équipement de secours.

Nous avons toutefois besoin de poser une hypothèse supplémentaire pour effectuer ce changement : comme les procédures sont indivisibles, nous perdons le fait qu'une panne puisse interrompre la procédure en cours. Il nous faut donc étendre l'hypothèse de non-interruption (Théorème 39 page 136) à une procédure, ce qui est effectué naturellement.

Théorème 9 (*Traduction en DTMDP avec procédures*)

On définit le processus décisionnel markovien suivant :

S l'espace d'états demeure identique : $M_0 \times \ldots \times M_N$ est l'ensemble de tous les modes possibles.

A l'espace des actions est l'espace de toutes les procédures possibles : en notant T_C l'ensemble des transitions contrôlables, A est l'ensemble ordonné des parties de T_C, soit $\mathbb{N}^{T_C}$.

$\mathring{R}$ la fonction de récompense où $\mathring{R}(s,a) = f(s_{|a})E(t(s_{|a}))$ telle que définie précédemment, où $s_{|a}$ est l'état obtenu immédiatement après l'exécution de l'action a dans l'état s et $E(t(s_{|a}))$ est le temps moyen (i.e. espérance) où le système restera dans l'état $s_{|a}$.

$\mathring{T}$ la fonction de transition, définie telle que dans $\mathring{T}(s,a,s')$, s' est le résultat de l'application de l'action a dans l'état s, suivie exactement d'une transition exogène ; $\mathring{T}$ a pour valeur : $\mathring{T}(s,a,s') = \frac{\sum_{e \in D_{s_{|a}}} \lambda_e}{\sum_{e \in D} \lambda_e}$ où D est l'ensemble de toutes les transitions exogènes pouvant survenir dans $s_{|a}$, et $D_{s_{|a}}$ est l'ensemble de toutes les transitions exogènes pouvant survenir dans $s_{|a}$ et menant à s'.

I est un état initial spécifié.

Le critère optimal demeure le même : la solution de ce MDP est une politique π maximisant la récompense espérée.

Pour effectuer ce passage d'un espace d'actions à un espace de procédures, nous avons deux possibilités : (1) nous effectuons ce passage à la volée dans l'algorithme de résolution, ce qui nécessite d'avoir un algorithme favorisant la recherche d'une succession d'actions avant de considérer les défaillances pouvant survenir, ou (2) nous effectuons un traitement préliminaire pour calculer explicitement l'espace des procédures utiles dans chacun des états de décision.

Bien que l'option (2) soit la plus générale, puisqu'elle est applicable à toutes les méthodes de résolution, nous avons choisi l'option (1) dans notre résolution puisqu'elle est compatible avec la méthode de résolution que nous avons utilisée dans le processus outillé final. De plus, l'option (2) a le défaut principal d'augmenter de façon importante l'espace des actions à explorer ; ceci peut être compensé par une résolution intermédiaire, par exemple au travers de méthodes de planification classique ou de résolution de contraintes, pour ne sélectionner dans l'espace des actions que les procédures qui améliorent les qualités de service ; cependant, le gain en termes de temps de résolution n'est pas garanti, et nécessite une adaptation propre à chaque cas d'application.

VI.2.6 Ajout de contraintes PCTL

Enfin, nous devons préciser comment se traduisent les critères et les contraintes que l'on souhaite imposer sur la qualité de service : en effet, au travers de la fonction de préférence présente dans la fonction de récompense, nous avons exprimé des critères portant sur les qualités de service instantanées à chaque temps où le système est dans un état de fonctionnement acceptable ; la seconde exigence correspond à des contraintes de sécurité, puisque l'on souhaite pouvoir exprimer que certains états non acceptables doivent être évités avec une certaine probabilité.

Comme nous l'avons justifié dans les chapitres précédents, ceci peut être effectué au travers de l'ajout de contraintes PCTL. L'ajout de contraintes PCTL à ce processus décisionnel markovien peut être directement effectué de la manière suivante :

Définition 40 (*Spécifications de sécurité*)

On définit $Req \subset S \times Q_S \times [0;1]$ l'ensemble des spécifications (Requirements) sur les services, où $Req(s, q_s, p)$ décrit le fait que le service s ne doit pas avoir une qualité de service plus faible que q_s, avec une probabilité p exprimée dans la même unité que les termes λ, depuis la configuration initiale.

Ceci signifie que nous considérons toutes les chaînes de transition menant à une configuration où le service s n'est pas fourni avec une qualité de service suffisante, puis que nous calculons la probabilité que chacune de ces chaînes se produise et faisons la somme de ces probabilités. Nous comparons alors cette probabilité avec p.

Dans le CTMDP, cette probabilité est directement la probabilité d'atteindre une configuration (un

état) où la qualité de service n'est pas suffisante ; ceci se traduit directement en contrainte PCTL $(true)U^{<p}g$, où g est une fonction booléenne étant vraie dans une configuration dès lors que s n'est pas fournie avec au moins une qualité de service q_s.

Cependant, dans le processus de saut, la probabilité de chaque transition n'est pas la probabilité qu'un évènement survienne dans un faible incrément de temps ; nous avons en effet supprimé toutes les considérations de temps lors de la construction du DTMDP, par conséquent la somme des produits des transitions dans une chaîne menant à un état particulier dans un DTMDP ne représente pas la probabilité d'atteindre cet état dans un faible incrément de temps, mais plutôt la probabilité de l'atteindre après un temps infini.

Il nous faut donc adapter la manière dont nous calculons chaque contrainte PCTL, en multipliant pour chaque transition la probabilité $q(s, a, s')$ que la transition arrive dans un faible incrément de temps, et non la probabilité $T(s, a, s')$. L'algorithme de calcul, basé sur l'algorithme Value Iteration, demeure le même.

De plus, il est possible de choisir une interprétation différente des contraintes de sureté : telles que nous les avons écrites, les contraintes peuvent porter sur les états de décision (juste après une défaillance), sur les états "nominaux" (juste après une décision) ou sur les deux types d'états. Dans le DTMDP, les états les plus accessibles sont les états après défaillance ; cependant, si nous souhaitons nous limiter à ces états pour les contraintes, alors certains états seront marqués comme incorrects alors qu'il existe une procédure permettant d'amener le système dans un état avec une qualité de service satisfaisante.

Pour cette raison, nous avons choisi dans nos cas d'application de **nous limiter aux états "nominaux" pour le calcul des contraintes** (Figure 27 page 139), c'est-à-dire aux états juste après une décision. Ceci ne modifie pas l'algorithme de calcul des contraintes PCTL, puisque sur une chaîne de Markov donnée, avec une politique fixée, il nous suffit de ne modifier les valeurs des fonctions booléennes g que sur les états correspondants.

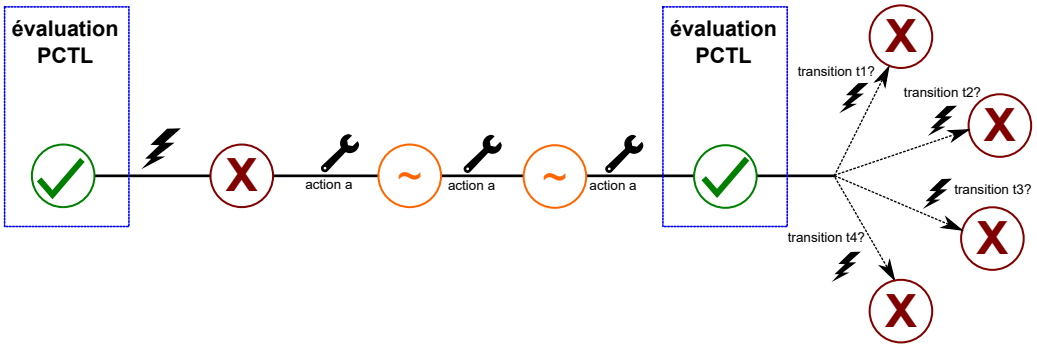


FIGURE 27 – Modèle GMD : les contraintes PCTL ne sont évaluées qu'aux états immédiatement après la dernière décision d'une procédure.

VI.3 Sémantique de la qualité de service

Dans les paragraphes précédents, nous avons défini de façon formelle les notions de ressource, mode, service, qualité de service et transition, en particulier en rattachant la dynamique du système à un modèle de processus décisionnel markovien. Cependant, cette définition formelle n'explique pas l'intention du modèle, c'est-à-dire les cas d'usage ainsi que le type de cas d'application que ce modèle est fait pour traiter.

VI.3.1 Catégorisation des qualités de service

Une approche naturelle qu'il est possible de suivre est celle de la catégorisation de la qualité de service ; il faut néanmoins noter que dans la littérature, le terme "qualité de service" renvoie le plus souvent au domaine de la gestion de flux réseaux, c'est-à-dire que la qualité de service représente une qualité de transmission de l'information, par exemple en termes de réactivité ou de débit. Des études de catégorisation de la qualité de service dans ce contexte ont été réalisées, telles que [EFJ+98], [HCM+00], [WC96] ou [GJS03]

En nous basant sur ces études et sur la connaissance du domaine avionique que nous avons évoquée précédemment, nous proposons la catégorisation suivante, sans prétention d'exhaustivité :

Catégorisation thématique de la qualité de service

- **Précision** : désigne dans quelle mesure le service est rendu avec du bruit, des erreurs de calcul ou de mesure, ou toute autre forme d'incertitude sur la variabilité des valeurs des données.
- **Disponibilité** : désigne dans quelle mesure le service peut être utilisé lorsque le système en a besoin, par exemple sous la forme d'un pourcentage de temps où le service est opérationnel ou d'une probabilité que le service puisse être utilisé.
- **Temps de réponse** ou temps de démarrage : désigne une estimation du temps nécessaire avant de pouvoir utiliser ce service.
- **Bande passante** : désigne une estimation de la quantité d'informations échangées par ce service, ou de la capacité d'échange d'informations, en termes de quantité, que ce service peut fournir.
- **Intégrité** : désigne la fiabilité des données, c'est-à-dire une estimation de confiance sur le fait que les données demeurent dans une certaine enveloppe autour des données qui auraient été émises en situation nominale, lorsque aucune défaillance n'est survenue.
- **Fiabilité** [a] : désigne la résilience du service à des défaillances futures, sous la forme d'un nombre de défaillances nécessaires, d'une probabilité de défaillance (de service) ou encore d'un niveau de DAL [b].
- **Sûreté** : désigne, au sens informatique, l'estimation de l'accès protégé aux données, c'est-à-dire d'un accès qui n'est pas corruptible par un attaquant extérieur ou qui au contraire a été corrompu dans une certaine mesure.
- **Synchronicité** : désigne la capacité ou la nécessité du service à être utilisé dans une communication directe ou dans une communication différée ; cette capacité peut être exprimée en termes de temps minimal ou maximal entre chaque requêtes, ou en capacité de mémoire tampon (buffer) sur le nombre de requêtes que le service peut accepter ou fournir, simultanément ou dans un temps court (relativement à la vitesse de traitement des requêtes).

a. reliability en anglais
b. Design Assurance Level [ARP 4761]

Nous avons désigné cette catégorisation comme "thématique" puisqu'elle classifie les qualités de

service selon le sens qui est donné à la valeur. Notons par ailleurs que nous n'avons pas donné explicitement de valeurs pour les domaines que peuvent prendre ces qualités de service : en effet, chaque cas d'application nécessitera un niveau de granularité différent dans les domaines, par exemple entre un cas d'application pour lequel il sera suffisant de distinguer une qualité de service de précision "peu précis" et "très précis", et un cas d'application pour lequel on souhaitera quantifier la précision sous la forme d'intervalles "+-10%", "+-20%",...

VI.3.2 Conséquences de la gestion de la qualité de service sur l'architecture du système

Une des conséquences implicite du modèle de gestion de modes dégradés est qu'il impose un certain nombre d'ajustements à l'architecture du système. Plus spécifiquement, il est nécessaire de disposer de méthodes permettant d'évaluer la qualité de service à certains points clés du système. Nous détaillons les cas de figure possibles dans la classification suivante :

Détection de changement de la qualité de service

Soit A fournissant un service à B. B peut détecter une défaillance d'un service fourni par A à partir de :
- la détection par **présence/absence** de signal : A ne fournit plus de service.
- la détection par **mesure directe** : B mesure directement la qualité du service fourni par A (exemple : mesure de la bande passante, synchronicité, temps de réponse, ...).
- la détection par **qualifieur** : B est informé par A de la défaillance d'un de ses services par un canal d'information supplémentaire (exemple : fiabilité, disponibilité, sécurité) ; ce changement de qualité de service est généralement causé par A suite à une défaillance (matérielle ou d'un de ses services d'entrée).
- la détection par **comparaison à une référence** : B détecte une défaillance du service fourni par A en comparant avec une ou plusieurs autres sources (exemple : précision).

Ceci nous montre par ailleurs une hypothèse que nous n'avons pas explicitée précédemment : dans la construction du modèle MDP, nous avons fait l'hypothèse d'une observabilité totale des qualités de service, ce qui implique que l'agent doit avoir un moyen d'obtenir ou d'inférer la qualité de tous les services ; ceci est immédiatement problématique dans certains cas, en particulier lorsque tous les systèmes recevant un même service ne sont pas ou plus capables de mesurer la qualité de ce service. Cette hypothèse est cependant rendue possible si nous choisissons de nous limiter à un nouveau système, c'est-à-dire un système tel que la mesure de la qualité de service en tout point peut être assurée grâce à l'ajout d'équipements, tels que des équipements d'alerte, de test et de diagnostic.

Si tel n'est pas le cas, et si les équipements de diagnostic ne permettent pas une observabilité totale des services, alors d'autres méthodes doivent être envisagées (par exemple les méthodes de résolution POMDP [PCCT13] ou planification conformante [CR00], ou il est tout du moins nécessaire de réaliser une étude supplémentaire pour s'assurer que les décisions prises par l'agent sont toujours possibles malgré ce manque d'information : si l'agent ne connaît par exemple pas la valeur de la qualité de service d'une sonde, mais qu'il n'a pas besoin de cette information pour effectuer une décision, alors la démarche que nous proposons est toujours valide. Si en revanche le fait d'avoir ou non cette information change potentiellement la décision de l'agent, alors il est nécessaire d'étudier si cela justifie l'ajout d'une mesure supplémentaire de mesure au niveau de la sonde (qui est une solution potentiellement coûteuse), ou si une logique particulière permet d'inférer la valeur de la qualité de service, ou bien encore si le fait d'avoir une décision non-optimale n'a qu'un impact mineur sur les critères de sécurité et les contraintes d'optimalité.

Dans les paragraphes suivants, nous nous appuierons sur ces réflexions pour évaluer dans quelle mesure les cadres mathématiques existants permettent de représenter des problèmes de Gestion de Modes Dégradés.

VI.3.3 Sélection d'un modèle mathématique

Nous avons déjà partiellement effectué le choix d'un modèle mathématique lors de l'expression de la dynamique du système : en effet, nous avons choisi de représenter cette dynamique à partir d'un processus décisionnel markovien, et d'utiliser la logique PCTL pour exprimer des contraintes probabilistes sur les qualités de service. Nous pouvons à présent nous poser la question de la légitimité de ce choix, et de ses conséquences : dans quelle mesure d'autres modèles mathématiques auraient pu être utilisés, et quelles sont les conséquences de ces deux choix sur les méthodes de résolution ?

Nous avons déjà évoqué précédemment différentes classes de modèles utilisés traditionnellement dans les problèmes de décision. En particulier, nous avons isolé trois catégories principales : l'optimisation sous contraintes [Lue73], la planification classique [GNT04] et la planification probabiliste [Put94].

Parmi ces trois catégories, l'optimisation sous contraintes est la seule à ne pas être, de façon native, dynamique : bien qu'il soit possible d'encoder la dynamique d'un problème sous la forme d'un ensemble de contraintes (puisqu'il est possible de résoudre des problèmes de planification sous la forme d'une résolution de contraintes), ce cadre se prête traditionnellement à des problèmes où certains paramètres fixes doivent être déterminés. Par exemple, il serait possible d'utiliser ce type de techniques pour déterminer un ensemble de modes à fixer tels que certaines qualités de service soient maximales, ou respectent certains critères ; en revanche, ceci ne prendrait pas en compte les évolutions futures possibles, c'est-à-dire les défaillances pouvant survenir. Dans la mesure ou nous voulons garder cette prise en compte des évènements redoutés dans notre résolution - puisque nous nous intéressons à la fois à des critères d'optimalité et des contraintes de sécurité - le cadre de l'optimisation sous contrainte n'est donc pas suffisant.

Le même constat peut être établi pour le cadre de la planification classique : puisque nous avons effectué le choix d'une dynamique probabiliste, les méthodes de résolution de la planification classique ne sont pas suffisantes pour notre problème (notons que dans ce contexte le terme de "planification classique" n'englobe pas les méthodes de planification/replanification classiques qui peuvent être utilisées pour résoudre des problèmes MDP). Si nous n'avions pas choisi une dynamique probabiliste, d'autres méthodes auraient pu être appliquées, mais qui auraient répondu à d'autres problèmes ; ainsi, nous aurions pu par exemple poser la question de l'absence complète d'observabilité, par exemple en cherchant s'il existe une procédure garantissant qu'on puisse amener le système dans un état "sûr" - ou tout du moins connu - quel que soit l'état initial. Ce type de problèmes aurait alors pu être résolu avec des méthodes de planification conformante [CR00].

Une autre hypothèse que nous aurions pu prendre est celle de l'observabilité partielle : si certaines qualités de service ne sont pas connues de l'agent, alors nous aurions dû nous orienter vers des méthodes de type POMDP ; ces méthodes ont des conditions particulières, puisque certaines actions sont alors prises non pas de façon à optimiser une qualité de service, mais de façon à déterminer dans quel état le système se trouve réellement. En faisant l'hypothèse de l'observabilité totale, nous faisons en réalité l'hypothèse que cet aspect de recherche de connaissance sur l'état du système est laissé à un système externe, de façon préliminaire à la décision ; ceci suppose en particulier que les outils de diagnostic sont parfaits, ou tout du moins parfaits sur l'ensemble des qualités de service importantes à la décision. Toutefois, il semble clair qu'un modèle basé sur l'observabilité partielle présente des caractéristiques très intéressantes, bien que nous ne l'ayons pas étudié dans nos travaux.

Enfin, il est possible de remettre en cause le choix de l'expression sous forme de contraintes PCTL pour les contraintes de sécurité. Plus spécifiquement, nous aurions pu choisir une expression de contraintes s'alliant de façon différente au modèle MDP, tel que ce qui peut être fait dans des modèles tels que CMDP [Alt99] ou les travaux de Etessami et al. [EKVY07]. Nous avions déjà évoqué dans l'état de l'art préliminaire que de nombreux modèles existent permettant d'allier MDP et contraintes sur les états ; ces modèles permettent donc d'optimiser la qualité de service, en étant robustes aux défaillances de ressources pouvant survenir, tout en évitant certaines situations. Nous avions conclu dans le cas de l'aide à la décision pour le business jet que seuls des modèles de type PCMDP avaient le niveau de

garantie que nous souhaitions en termes de respect des contraintes de sécurité, puisque dans les autres modèles il était possible d'affecter (uniquement) des préférences telles que le système trouvait qu'il était plus avantageux de passer outre ces contraintes ; dans les modèles de gestion de modes dégradés, nous avons cependant une relation forte entre les contraintes de sureté et les contraintes d'optimalité : dans presque tous les cas, la solution optimisant les contraintes d'optimalité respectera naturellement les contraintes de sécurité, puisque l'optimalité correspond en partie à maintenir l'avion en vol. De plus, la plupart des modèles ne présenteront aucune boucle dans le processus décisionnel markovien, puisque toutes les reconfigurations prises par l'agent en réaction à des défaillances seront faites pour mitiger une défaillance venant de survenir. Par conséquent, dans presque tous les cas rencontrés réellement dans le contexte industriel qui est le nôtre, il suffira d'effectuer une résolution du MDP puis de vérifier le respect des contraintes sur la solution obtenue après-coup.

Néanmoins, dans le cas général, il est possible de construire des modèles à partir du formalisme GMD qui ne peuvent être exprimés que sous la forme d'un modèle de type PCMDP. C'est en particulier le cas lorsque les critères d'optimalité et les contraintes de sécurité sont opposées, par exemple si le pilote souhaite très fortement avoir une précision élevée sur sa position, mais que reconfigurer l'avion dans ce sens pourrait potentiellement mener à une situation critique si une nouvelle défaillance survenait. Bien que de tels cas ne soient que théoriques - puisque notre expérience nous a montré que de telles situations ne semblaient pas se produire dans la réalité - le fait qu'ils existent nous force à considérer une résolution centrée autours du modèle PCMDP complet. Il est important de noter que, contrairement à ce que nous avions pu effectuer dans les chapitres précédents, le modèle PCMDP que nous construisons à partir d'un formalisme GMD nécessite des contraintes PCTL non limitées à des probabilités 0 ou 1 ;

VI.4 Conclusion sur la modélisation de scénarios de décision dans le contexte avionique

Dans les chapitres précédents, nous avions construit un processus outillé permettant d'assister les décisions de maintenance dans un contexte avionique ; en particulier, nous avions bâti ce processus autour du modèle SPC MDP, permettant de prendre en compte des critères d'optimalité et des contraintes de sécurité.

Dans ce chapitre, nous avons donc étendu notre réflexion à d'autres problématiques de décision, afin d'étudier dans quelle mesure les caractéristiques que nous avions identifiées précédemment sont aussi applicables à d'autres problèmes. Pour atteindre cet objectif, nous avons donc réalisé les apports suivants :

- Au travers d'une étude des considérations de décision dans un contexte avionique, nous avons donné une description informelle de la **notion de qualité de service**, qui est une notion centrale dans de nombreux problèmes de décision.
- Nous avons alors défini un nouveau formalisme, le **modèle de Gestion de Modes Dégradés**, centré sur la notion de qualité de service et sur la propagation de la défaillance de cette qualité dans un système.
- Nous avons enfin montré que le cadre PCMDP était le mieux à même de représenter la totalité du modèle formel.

Cependant, les hypothèses que nous avions pu prendre précédemment pour simplifier le modèle PCMDP ne sont plus applicables dans le cas du modèle de Gestion de Modes Dégradés - plus spécifiquement, il n'est plus possible de se limiter aux seules contraintes saturées. Par conséquent, il nous faut à présent développer une nouvelle méthode permettant de faire face à la complexité, en termes de temps de résolution, du modèle PCMDP.

Dans le chapitre suivant, nous définirons un nouvel algorithme de résolution pour les modèles PCMDP, inspiré de différentes techniques de maîtrise de la complexité qui sont utilisées dans la littérature existante par des algorithmes de planification performants.

CHAPITRE VII

DÉVELOPPEMENT D'UN ALGORITHME DE RECHERCHE HEURISTIQUE PERMETTANT DE TRAITER LE PROBLÈME DE GESTION DES MODES DÉGRADÉS

Sommaire

DANS le chapitre précédent, nous avons élargi notre cadre de réflexion à plusieurs cas de problèmes de conception sûre et optimale dans le cadre avionique ; nous avons en particulier identifié que la plupart de ces problèmes traitaient de la gestion de la capacité du système à effectuer certaines tâches. Nous avons alors formalisé cette notion au travers d'un modèle de gestion de modes dégradés, centré sur la propagation de la qualité de service dans un système, et montré que plusieurs informations pouvaient être obtenues à partir de l'exploitation de ce modèle, en particulier sous l'angle d'un PCMDP.

Cependant, nous avons établi que le modèle SPC MDP, que nous avions utilisé pour résoudre de manière plus efficace certains problèmes PCMDP, ne pouvait pas être appliqué dans ce nouveau cas : les contraintes utilisées nécessitent de prendre en compte des probabilités autre que 0 et 1. La résolution de PCMDP à partir des algorithmes disponibles dans la littérature n'étant pas envisageable pour un système industriel, en raison de la complexité en temps de calcul, il nous faut donc à présent étudier dans quelle mesure la performance de résolution des problèmes PCMDP peut être améliorée par un autre algorithme.

Notre objectif dans ce chapitre est ainsi de développer un algorithme de résolution pour PCMDP permettant de traiter des modèles de grande taille, et qui soit compatible avec le modèle de gestion de modes dégradés.

Pour cela, nous allons dans un premier temps définir un ensemble d'hypothèses que nous ajouterons au modèle PCMDP pour permettre une résolution plus efficace. Dans un second temps, nous formaliserons un algorithme de résolution basé sur l'exploration à la volée de l'espace d'états, à l'aide d'heuristiques sur les solutions valides ; nous prouverons la validité de cet algorithme et détaillerons les méthodes d'ajustement des paramètres permettant d'obtenir une résolution efficace. Enfin, nous évaluerons les performances de ce nouvel algorithme sur différents cas de tests.

VII.1 Étude des concepts de recherche heuristique

Comme nous l'avons détaillé dans un chapitre précédent, le modèle PCMDP est construit sur le modèle MDP. Plus précisément, nous nous basons sur les hypothèses de Processus Décisionnels Markoviens Dévalués à Horizon-Infini (Infinite-Horizon Discounted-Reward (IHDR) MDP [Put94]) que nous rappelons ci-après :

Définition 41 (*IHDR MDP (rappel)*)

Un IHDR MDP est un vecteur $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, s_0, \gamma \rangle$*, où :*
- $\mathcal{S}$ *est l'espace des états.*
- $\mathcal{A}$ *est l'espace des actions.*
- $\mathcal{R}(s,a)$ *est la fonction de récompense, donnant une récompense pour l'action* a *dans l'état* s*.*
- $\mathcal{T}(s,a,s')$ *est la fonction de transition, donnant une probabilité d'atteindre l'état* s' *lorsqu'on effectue l'action* a *en* s*.*
- $s_0 \in S$ *est un état initial.*
- $\gamma \in [0,1)$ *est un facteur de dévaluation.*

De la même manière que précédemment, nous prenons comme hypothèse que tous les composants de ce MDP sont connus, et que $\mathcal{S}$ et $\mathcal{A}$ sont finis.

Pour exprimer les contraintes sur la politique du PCMDP, nous nous baserons à nouveau sur le langage *Probabilistic Real Time Computation Tree Logic* (PCTL) [HJ94].

Définition 42 (*Contraintes PCTL (rappel)*)

Les contraintes PCTL sont exprimées à l'aide de l'opérateur temporel probabiliste "Strong Until" $fU^{\leq H}_{\diamond p}g$*, où :*
- $\diamond$ *est un opérateur de comparaison (*$<$,$\leq$,$\geq$, *ou* $>$*).*
- f *et* g *sont des fonctions booléennes sur les états.*
- $H \in \mathbb{N}$ *est un horizon.*
- $p \in [0,1]$ *est une probabilité.*

Pour un chemin d'exécution aléatoire d'une politique donnée, la formule $fU^{\leq H}g$ *doit être vraie avec une probabilité au moins/au plus/égale à* p*.*

La formule en elle-même est vraie lorsque f *demeure vraie pour tous les états jusqu'à ce que* g *devienne vraie.* f *et* g *peuvent, mais n'y sont pas obligées, demeurer vraies par la suite. De plus,* g *doit devenir vraie au plus après* H *étapes après le début de l'exécution de la politique. Si* H *est égal à l'infini, alors* g *doit devenir vraie à un certain point du chemin d'exécution.*

Nous nous intéressons par exemple à ce type de contraintes PCTL (l'état initial s_0 est implicite) :
- $(true)U^{\infty}_{>0.8}g$: lorsqu'on applique la politique, la probabilité d'atteindre un état où g est vraie doit être au moins 0.8 (contrainte d'atteignabilité).
- $(true)U^{\infty}_{<0.2}g$: lorsqu'on applique la politique, la probabilité d'atteindre un état où g est vraie doit être au plus de 0.2 (contrainte d'évitement).

Par exemple, si l'on souhaite modéliser un système de contrôle d'une centrale électrique, on pourrait souhaiter assurer qu'un mode "sécurisé" puisse être atteint avec une probabilité d'au moins 90%. Puisque plusieurs actions peuvent être nécessaires pour atteindre un tel état, et que de nombreuses défaillances peuvent survenir durant le chemin, c'est en effet une contrainte de chemin. Celle-ci peut être exprimée sous la forme : $(true)U^{\infty}_{>0.9}g$, où g est une fonction booléenne qui est vraie seulement dans l'état sécurisé. Notons que g ne représente pas nécessairement un état absorbant, tel qu'un but ou une impasse.

L'un des avantages de cette approche est qu'une telle fonction g peut être facilement exprimée à partir de certaines variables pertinentes. Par exemple, g pourrait être basée sur un niveau d'énergie de sortie, signifiant que le système est considéré dans un état "sécurisé" dès lors que son énergie de sortie est plus faible qu'un certain seuil fixé.

Il est important de noter la distinction majeure entre le fait de fixer des récompenses ou des pénalités (contraintes faibles) et de s'assurer que les contraintes PCTL sont garanties (contraintes fortes). En effet, bien qu'il soit possible d'utiliser des récompenses avec une forte valeur pour guider l'agent vers des objectifs particuliers, et des récompenses $-\infty$ pour interdire certains états, un tel encodage n'est plus

possible lorsqu'il s'agit d'exprimer des contraintes PCTL : avec des récompenses $-\infty$, un algorithme IHDR MDP ne s'arrêterait jamais dans le cas où les états interdits sont inévitables ; de plus, dans un PCMDP le système est libre de continuer à accumuler des récompenses même après la complétion des objectifs obligatoires - ce qui n'est pas le cas d'autres cadres mathématiques classiques tels que SSP MDP.

VII.1.1 Étude des hypothèses sur la forme des contraintes et la forme des solutions

Lors de la définition du modèle SPC MDP, nous avions posé un certain nombre d'hypothèses, en particulier sur les contraintes PCTL utilisées, qui nous ont permis d'obtenir une résolution plus efficace que les algorithmes existants. De façon similaire, la première étape du développement de ce nouvel algorithme est de détailler les hypothèses que nous imposons au modèle PCMDP.

Contraintes transitoires

L'une des hypothèses que nous posons sur les contraintes PCTL est que toutes les contraintes sont transitoires :

Définition 43 (*Contraintes transitoires*)

Une contrainte encodée par une paire de fonctions booléennes $(f, g) : \mathcal{S} \to \{true, false\}$ est ***transitoire*** *si les trois ensembles où $(f(s), g(s)) = (true, true)$, $(false, false)$, $(false, true)$, respectivement, sont absorbants.*

Cette définition est une définition alternative de la définition des fonctions transitoires établie lors d'un chapitre précédent (Définition 29 page 86).

Cette condition nous dit simplement que quelque chose dans l'espace d'états doit enregistrer de façons différentes lorsqu'un état où g est vrai a été atteint, et lorsque la contrainte a été définitivement perdue ; la situation correspondante doit alors être absorbante, c'est-à-dire que l'espace d'états est séparé en quatre parties, dont trois sont isolées des autres. Cette condition n'est pas nécessaire pour le fonctionnement de l'algorithme, mais il s'agit d'une condition qui est logique dès lors qu'on cherche à obtenir une politique sans mémoire pour un PCMDP, en raison des explications que nous avons établies lors de la construction de SPC MDP : la politique solution devrait agir différemment lorsque le but a été atteint, car sinon elle chercherait à continuer d'essayer de valider le but - atteindre un état où g est vraie - alors même qu'il a déjà été validé.

Dans notre algorithme, nous nous basons sur l'hypothèse que toutes les contraintes sont transitoires. Notons que même sans cette condition, l'algorithme parviendrait toujours à trouver une politique déterministe optimale valide, mais cette politique présenterait des décisions illogiques puisqu'elle ne prendrait pas en compte le fait qu'une contrainte a été validée.

À nouveau, il est très facile d'adapter les contraintes non transitoires à des contraintes transitoires, au travers de l'ajout d'une variable mémoire à l'espace d'états, qui peut être réalisé à la volée (Théorème 6 page 86).

Horizon fini sur les contraintes

Une autre hypothèse classique que nous pouvons prendre est celle de supposer que toutes les contraintes ont un horizon infini. En effet, une contrainte ayant un horizon fini h représente simplement le fait que la fonction g doit être vraie avant h étapes ; cette contrainte peut être exprimée autrement si on construit une nouvelle fonction g' identique en tout point à g sauf qu'elle ne peut jamais être vraie après h étapes depuis l'état initial. Cette transformation peut être réalisée avant le début de l'algorithme en ajoutant à l'espace d'états une variable mémoire contenant le nombre d'étapes depuis l'état initial, et en arrêtant de compter au maximum de tous les h_ξ ; Les fonctions g_ξ peuvent être adaptées pour ne jamais être vraies une fois que h a été dépassé.

Ceci a deux avantages : (1) l'algorithme devient plus lisible, et (2) ceci nous permet de considérer que les politiques solutions sont sans mémoire, bien que la partie "mémoire" soit en réalité gérée au sein même de l'espace d'états. Si certaines contraintes ont des horizons finis, il est logique que le nombre d'étapes soit une variable qui soit prise en compte dans les politiques (au travers de l'espace d'états ou

parce que la politique est à mémoire), puisque le système ne devrait pas agir de la même façon selon qu'il atteint un *état but* après un faible ou un grand nombre d'étapes ; comme la conversion entre les deux approches est facile à mettre en œuvre (Théorème 4 page 78), nous choisissons à nouveau pour notre algorithme la solution de l'ajout à l'espace d'états, en nous basant sur les mêmes arguments que ceux détaillés lors de la construction de SPC MDP.

Dans les paragraphes suivants, lorsque nous écrirons une contrainte PCTL nous omettrons par conséquent la partie ∞ pour les horizons des contraintes.

Path-Constrained Markov Decision Processes

Nous pouvons alors rappeler la définition d'un PCMDP [TK12a] :

Définition 44 (*PCMDP (rappel)*)

Un processus décisionnel markovien à chemin contraint (PCMDP) est un vecteur $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, s_0, \gamma, \xi \rangle$, *où* $\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, s_0$ *et* γ *correspondent à la définition d'un IHDR MDP, et* $\xi = \{\xi_1, \ldots, \xi_n\}$ *est un ensemble de contraintes PCTL. Pour chaque* $i \in [1, n]$, $\xi_i = f_i U^{\infty}_{\diamond p} g_i$.

Une politique d'un PCMDP π *est dite* valide *si elle satisfait à toutes les contraintes. Une* solution optimale *est une politique* π *qui est valide et a la meilleure valeur à l'état initial* $V^{\pi}(s_0)$ *de toutes les politiques valides :*

$$\forall \text{ valide } \pi', V^{\pi}(s_0) \geq V^{\pi'}(s_0).$$

Forme des solutions

Nous avons prouvé précédemment [SKTK14] que la solution optimale d'un PCMDP est potentiellement une politique aléatoire, c'est-à-dire une politique $\pi : \mathcal{S} \times \mathcal{A} \to [0; 1]$ par opposition à une politique déterministe $\pi : \mathcal{S} \to \mathcal{A}$. En effet, un exemple simple nous a permis de montrer que certaines politiques déterministes valides pouvaient ne jamais exister (Figure 14 page 81).

Cependant, nous choisissons à présent comme hypothèse de chercher explicitement des politiques déterministes solutions : notre but est de trouver une politique déterministe π qui soit valide et optimale parmi les politiques déterministes valides :

$$\forall \text{ valide et déterministe } \pi', V^{\pi}(s_0) \geq V^{\pi'}(s_0).$$

Ceci a pour implication que nous pouvons ne pas trouver de solution, si la combinaison de contraintes est telle qu'elle nécessite une politique aléatoire ; il peut par ailleurs aussi exister des politiques aléatoires qui ont une meilleure valeur que notre politique déterministe solution. Notons que ce type de restriction à des politiques déterministes a déjà été étudié sur d'autres modèles que PCMDP, par exemple sur le modèle CMDP [CF07].

Néanmoins, dans la plupart des cas les politiques déterministes sont plus adaptées que les politiques aléatoires lorsqu'il s'agit de les mettre en œuvre dans des systèmes réels : comme nous l'avons vu précédemment, un des inconvénients des politiques aléatoires est que nous devons générer un choix aléatoirement à chaque étape de décision, ce qui est contre-intuitif pour des utilisateurs qui souhaitent s'assurer que le système reste dans des bornes prévisibles, par exemple au travers de contraintes PCTL.

La motivation principale du choix des politiques déterministes est que l'espace des politiques qu'il est nécessaire d'explorer se trouve considérablement réduit : l'ensemble des politiques aléatoires est continu, tandis que celui des politiques déterministes est fini. Lors de la construction du modèle SPC MDP, nous avions pu de façon similaire réduire notre espace de recherche, en nous concentrant sur des politiques particulières, les ω-politiques ; ce choix était en partie basé sur l'exemple simple du PCMDP à deux états (Figure 13 page 80). Comme nous souhaitons explorer une autre voie, ne nécessitant pas de devoir restreindre les contraintes PCTL, nous ne pouvons pas utiliser à nouveau ces ω-politiques, dont la validité repose sur l'hypothèse de saturation des contraintes. À la place, nous choisissons la classe des politiques déterministes.

Bien que cette hypothèse nous prive potentiellement de la solution optimale, elle ouvre de nouvelles possibilités pour la construction d'un algorithme de résolution : comme nous le verrons par la suite, il est possible d'utiliser des techniques d'exploration de l'espace des politiques telles que les techniques de Branch and Bound pour isoler les politiques valides et déterminer la politique déterministe optimale.

VII.1.2 Justification du choix d'un algorithme d'exploration des politiques partielles

L'algorithme que nous proposons est librement inspiré de la classe des algorithmes dits **Branch and Bound**. Il s'agit d'une classe d'algorithme faisant partie de la classe plus générale des **algorithmes de recherche dans les espaces d'états** : dans ce type d'algorithme, on dispose d'un espace de recherche S dans lequel on souhaite isoler l'élément maximisant une fonction donnée. Pour cela, on explore l'espace d'états S d'une manière particulière, permettant d'isoler de plus en plus précisément l'élément optimal. De nombreux algorithmes font partie de cette classe, depuis les plus simples "Deep-First-Search" et "Breadth-First-Search" aux plus complexes "Descente de Gradient" ou "A*". Parmi les autres algorithmes que nous aurions pu essayer d'adapter, nous pouvons par exemple citer les algorithmes de *recherche locale*, de *recuit simulé* ou encore les *algorithmes génétiques*, bien que ces algorithmes soient sensibles au problème de la présence d'optimaux locaux, qui est vraisemblablement présent dans notre cas comme nous le verrons par la suite.

Détaillons plus en avant l'algorithme Branch-and-Bound : le principe est de diviser l'espace d'états de façon récursive en des espaces plus petits, puis de maximiser la fonction d'optimisation sur ces ensembles. Cette séparation est le "branchement". Si on se limitait au branchement uniquement, cela reviendrait à une résolution par force brute, c'est-à-dire à tester individuellement tous les candidats. L'algorithme de Branch-and-bound améliore les performances en gardant en mémoire des "bornes" sur la valeur minimale et maximale qu'il essaie de trouver, puis en utilisant ces bornes pour supprimer des sous-ensembles. Pour calculer cette borne, on utilise généralement une fonction permettant d'estimer la valeur d'un sous-ensemble, appelée **heuristique**.

La mise en œuvre de cet algorithme est donc simple : il nous suffit de formuler notre problème sous la forme d'un problème de recherche, puis de disposer d'une heuristique satisfaisante permettant d'éviter une partie de l'exploration.

Dans notre problème, nous souhaitons trouver une politique optimale ; l'espace de recherche est donc l'espace des politiques. Nous voyons ici que le choix de se limiter à des politiques déterministes est salvateur, puisque cet espace des politiques est alors fini. Cependant, il nous faut associer une manière de naviguer à cet espace des politiques ; ceci peut être effectué de deux manières :

1. On passe d'une politique à une autre en **modifiant certaines des actions choisies**.
2. On regroupe les politiques en **sous-ensembles** en fonction de caractéristiques communes.

Pour la première option, il serait par exemple possible de choisir de modifier une politique à la fois ; cependant ceci a l'inconvénient principal qu'il est nécessaire d'établir une règle pour savoir quelle action choisir : on souhaite par exemple remplacer une action par une action améliorant la valeur de la politique, mais une telle action peut avoir comme états d'arrivée des états qui n'étaient pas présents dans la politique d'origine, nécessitant donc de choisir non plus une seule action, mais potentiellement tout un ensemble de nouvelles actions. Il est alors évident que cette solution est plus adaptée aux algorithmes de type "recherche locale", et que l'une des difficultés principales serait de gérer le fait qu'un changement faible d'action peut potentiellement entraîner un changement important dans l'optimalité ou la validité de la politique.

Pour la seconde option, il nous faut trouver une manière de regrouper les politiques en sous-ensembles, de préférence en sous-ensembles récursifs permettant ainsi d'appliquer une méthode Branch-and-Bound. La manière qui semble la plus naturelle pour cela est de les regrouper selon les décisions en partant de l'état initial : sur un chemin d'exécution unique, on souhaiterait dire que deux politiques ayant les mêmes décisions tout au long du chemin mais une décision différente à la dernière étape sont dans un même sous-ensemble ; ce sous-ensemble est contenu dans un ensemble plus grand contenant toutes les politiques ayant les même décisions jusqu'à l'étape $N-2$, et ce jusqu'à l'ensemble principal contenant toutes les politiques.

Cette notion recouvre en vérité l'idée de politique partielle : par cette intuition de "chemin unique jusqu'à l'étape N", on ne décrit pas en réalité un chemin, mais une manière d'explorer successivement des décisions en partant de l'état initial.

Dans les paragraphes suivants, nous détaillons plus en avant les deux notions nécessaires pour construire l'algorithme de Branch-and-Bound : la notion de politique partielle, permettant de structurer l'espace de recherche, puis la notion d'heuristique, permettant d'obtenir des résultats sur les ensembles de politiques partielles sans qu'il soit nécessaire de calculer la valeur ou la validité de toutes les politiques contenues dans cet ensemble.

Définition de la notion de politique partielle

Comme nous l'avons évoqué précédemment, l'algorithme est construit à partir de la méthode Branch-and-Bound : l'objectif est d'explorer méthodiquement l'espace de toutes les politiques, en gagnant à chaque étape de cette exploration des informations supplémentaires nous permettant d'éviter une partie importante de l'espace des politiques. Nous allons par conséquent définir une politique partielle comme un ensemble P de paires (état, action) : une politique partielle contient certains états s ayant une action fixée $P(s)$, et d'autres états pour lesquels aucune action n'a pour l'instant été fixée. De manière plus précise, ces autres états sont appelés les *états non-étendus*, et un état est dans l'ensemble des états non-étendus seulement s'il peut être atteint depuis l'état initial uniquement avec les actions qui ont déjà été fixées.

Définition 45 (*Politique partielle*)

Soit S un espace d'états, A un espace d'actions et s_0 un état initial. On définit une politique partielle P comme une fonction partielle de l'espace des états S dans l'espace des actions A.
On définit plus précisément, à titre de notation, une politique partielle P comme étant composée des éléments suivants :

- *une fonction $P : S \to A$ où $P(s)$ est l'action choisie dans l'état s.*
- *un ensemble $P.U \subset S$ des états non-étendus.*
- *une valeur $P.value$ qui est l'espérance de la récompense dévaluée totale depuis l'état initial s_0.*

Avec cette définition, il est à présent possible d'étendre une politique partielle donnée en prenant un état depuis son ensemble des états non-étendus, puis en choisissant une action pour cet état. À partir d'une politique partielle, on créera ainsi autant de politiques partielles "filles" qu'il y a d'actions possibles pour un état non-étendu donné.

Ceci nous montre bien que les politiques partielles sont un outil adapté pour le parcours de l'espace des politiques : en partant de l'état initial, il est possible de construire facilement un arbre des politiques partielles en assignant successivement des actions aux états non-étendus. Une fois que la politique partielle ne contient plus aucun état non-étendu, on obtient alors une politique complète. En obtenant des informations sur une politique partielle, nous pouvons alors déduire des informations potentielles sur toutes ses politiques filles, c'est-à-dire sur toutes les politiques partielles crées à partir de l'extension de la politique partielle mère.

Comme nous le verrons par la suite, il est ainsi possible d'estimer si une politique partielle est valide ou non. Par conséquent, il est possible de n'étendre que les politiques partielles valides, c'est-à-dire à partir desquelles il est possible de valider toutes les contraintes. Lorsqu'une politique partielle est prouvée invalide, nous savons que toutes ses politiques filles le sont aussi, nous évitant ainsi l'exploration de tout cet ensemble de l'espace de recherche.

Enfin, il est possible de comparer la valeur de deux politiques partielles, au travers d'une fonction que nous détaillerons. Ceci nous permet de choisir d'étendre en priorité les politiques partielles les plus prometteuses et, dès lors que nous avons trouvé une solution complète, nous pouvons restreindre notre exploration aux politiques partielles qui ont une valeur au moins aussi grande que la meilleure politique complète (i.e. sans états non-étendus) trouvée pour l'instant. Comme nous le verrons, ceci n'est correct que si la valeur des politiques partielles est plus grande que la valeur de ses politiques complètes filles, ce qui est obtenu grâce à des propriétés particulières.

Néanmoins, cette évaluation de validité ou de valeur est rendue complexe par le fait que certains états des politiques partielles ne sont pas étendus. Comme nous l'avons évoqué dans les paragraphes précédents, la plupart des algorithmes existants obtiennent ces données en exploitant une connaissance supplémentaire appelée heuristique.

Définition de la notion d'heuristique

De manière générale, une **heuristique** [Bon01] est une méthode permettant d'obtenir une solution pratique (et rapide) à un problème, sans garantie d'optimalité. Dans le contexte informatique, on parle plus précisément d'heuristique pour désigner une méthode de recherche utilisant des approximations pour obtenir des résultats plus rapidement qu'une méthode optimale.

Prenons l'exemple d'un labyrinthe : si un agent souhaite trouver la sortie d'un labyrinthe, la pire solution revient à explorer tous les chemins possibles ; il s'agit d'une résolution par force brute. Ceci peut par exemple être effectué, de façon pratique, en suivant toujours le mur de droite, ce qui correspond à un parcours en profondeur d'abord (Deep-First Search ou DFS). Dans le pire cas, cette méthode explore tous les segments de chemin, on parle alors d'une complexité dans le pire cas de $O(|E|)$ si E désigne l'ensemble des segments de chemin (c'est-à-dire un couloir entre deux intersections).

Supposons alors qu'on dispose d'une information supplémentaire, par exemple sur le fait que la sortie du labyrinthe se trouve au nord. Alors une meilleure stratégie en pratique est de choisir de préférence les chemins qui vont vers le nord à chacune des intersections. Cette information supplémentaire est appelée une heuristique.

Cependant, on se rend compte qu'un tel algorithme ne garantit pas de trouver une solution plus vite : la complexité dans le pire est cas est toujours la même, puisqu'on peut imaginer un labyrinthe pour lequel la première intersection au nord est en fait le pire choix possible. Pour que la solution puisse être trouvée plus rapidement, il nous faut disposer d'une information la plus précise possible : si par exemple on disposait à chaque intersection d'un panneau décrivant la distance exacte entre chacun des chemins possibles et la sortie, alors le problème serait trivial puisqu'il suffirait de choisir à chaque étape le chemin le plus court ; si en revanche les informations présentes sur le panneau sont une estimation de la distance qui n'est pas exacte, mais très proche, alors le temps de résolution peut se trouver augmenté, selon que cette information est précise ou très loin de la vérité. En pratique, deux critères sont déterminants pour cette information :

- Si un des panneaux indique une distance de 500 mètres pour un des chemins, mais que l'on sait que le chemin peut être atteint en 300 mètres, alors on peut se dispenser d'explorer le chemin de 500 mètres *dans la mesure où l'information ne surestime pas la distance* ; si en réalité le chemin estimé à 500 mètres est à une distance de 200, alors en n'explorant pas ce chemin on se prive de la solution optimale. Cette notion de non surestimation est la notion d'**admissibilité**.
- Plus l'information est proche de l'information maximale, c'est-à-dire d'un panneau nous disant la distance réelle de chaque chemin, plus on peut garantir de trouver une solution en un temps minimal.

L'algorithme A*, qui peut être vu comme un cas spécial de Branch-and-Bound, est l'algorithme le plus connu mettant en oeuvre ces principes : en partant d'un état initial, cet algorithme maintient en mémoire un ensemble d'états visités ainsi qu'un ensemble d'états à explorer ; l'ensemble des états à explorer est trié en fonction d'une priorité, c'est-à-dire que l'on place en tête de liste l'état avec la plus faible valeur d'heuristique. Cette valeur résulte de la somme de la fonction d'heuristique pour cet état - autrement dit l'estimation de la distance au but - et du coût pour atteindre l'état.

De façon pratique, une heuristique est souvent obtenue à partir de la résolution d'une version simplifiée du problème : dans le cas du labyrinthe, la simplification peut par exemple être effectuée sous la forme des quatre directions Nord/Sud/Est/Ouest. Une autre simplification peut être faite si on divise le labyrinthe en N zones arbitraires, que l'on marque les connexions possibles entre chaque zone, ce qui permet de n'explorer que les zones qui sont traversées par la solution finale ; plus N est grand et plus l'heuristique est informative, puisqu'elle est proche de la réalité ; cependant, plus N est grand et plus le coût de calcul de l'heuristique est important, jusqu'à être égale au temps de résolution du problème complet.

Dans la plupart des cas, la simplification peut-être effectuée en omettant certaines variables du

problèmes ; on peut par exemple imaginer que pour le guidage d'un robot martien, on considère en première approximation que la rugosité du sol n'est pas primordiale pour trouver le chemin le plus court, ce qui nous donne une assez bonne approximation de la trajectoire optimale. On parle **d'état relâché** pour désigner un état où certaines variables ont été ignorées. La sélection de quelles variables ignorer ou non est critique, ce qui implique notamment qu'un solveur spécialisé dans la résolution d'un seul problème est souvent bien plus performant qu'un solveur général, pouvant résoudre plusieurs domaines différents.

Pour pouvoir raisonner sur des états relâchés, il est nécessaire de décomposer les états en variables ; cependant, la manière dont cette décomposition est effectuée a un impact direct sur la possibilité de construire ou non des états relâchés. Comme nous l'avons déjà évoqué dans l'état de l'art préliminaire, une manière classique de représenter des états est sous le formalisme STRIP, où chaque état est composé d'un ensemble de **fluents**, c'est-à-dire des éléments atomiques booléens ; par exemple, on peut représenter un ensemble de cubes posés sur une table sous la forme de fluents "(on-table cube1), (on-table cube2), (on cube3 cube2)". Effectuer une action ou une transition revient alors à ajouter ou retirer certains fluents, par exemple retirer le fluent "(on cube3 cube2)" et ajouter le fluent "(on-table cube3)" pour signifier l'action de prendre le cube 3 et de le poser sur la table.

À partir d'une telle représentation, un état relâché peut être obtenu en ignorant tous les effets de suppression de fluents. En pratique, ceci nous permet d'aller plus rapidement au but, puisque l'ensemble des fluents d'un état ne peut que croître. Il est possible de montrer [Bon01] que cette simplification produit bien une heuristique admissible, puisque le problème relâché produit une borne inférieure du coût optimal - c'est-à-dire qu'elle est optimiste par rapport à l'heuristique parfaite.

Cependant, cette relaxation ne suffit pas, puisqu'il est aussi possible de montrer que le problème relâché a une complexité tout aussi grande que le problème d'origine (NP-difficile pour la planification classique). Plutôt que de chercher explicitement à atteindre le but dans le problème relâché, nous pouvons chercher à estimer la difficulté d'atteindre le but, par exemple en estimant la difficulté d'obtenir les fluents constituant le but.

Ceci est à la base de l'heuristique h_{add}, qui permet de calculer la difficulté d'atteindre un but comme étant la difficulté d'atteindre un ensemble de sous-buts, c'est-à-dire la difficulté d'établir les fluents individuels. Cette heuristique est dite *additive* puisqu'elle effectue une somme sur la difficulté des sous-buts, mais ce qui suppose aussi que les sous-buts sont indépendants, ce qui n'est pas le cas dans le cas général : en effet, il est tout à fait possible qu'atteindre un des sous-buts rende plus ou moins difficile de remplir un autre sous-but. Par conséquent, cette heuristique n'est pas admissible.

Une autre façon de combiner la difficulté des sous-buts est au travers d'un maximum : dans l'heuristique h_{max}, la difficulté d'un but est le maximum de la difficulté des sous-buts nécessaires pour l'atteindre, et récursivement. Cette heuristique est admissible, puisque le coût pour obtenir un ensemble de fluents n'est jamais plus bas que le coût pour obtenir chacun des fluents individuellement ; cependant, cette heuristique est beaucoup moins informative, puisqu'elle se concentre sur le sous-but le plus difficile à atteindre, en ignorant tous les autres.

Il existe cependant des méthodes pour améliorer la connaissance apportée par une heuristique ; l'algorithme GRAPHPLAN est par exemple l'un des algorithmes fondamentaux pour la recherche heuristique, puisqu'il a mis en avant l'utilisation de structures différentes pour la construction d'heuristiques efficaces. Dans cet algorithme, un graphe est construit sous la forme de couches successives représentant des fluents qu'il est possible d'établir au bout de N actions. Cette construction est appuyée par la notion de *mutex*, c'est-à-dire qu'on note certaines incompatibilités entre actions, par exemple des actions tendant à défaire ce qui a été établi précédemment ou à défaire ce qui est fait par une autre action de la même couche. À partir de ces incompatibilités, on peut alors poursuivre la construction des couches un peu plus loin que ce qui aurait été fait avec d'autres heuristiques, jusqu'à ce que toutes les incompatibilités aient été résolues. Les idées développées dans l'algorithme GRAPHPLAN ont permis de développer de nombreux algorithmes de recherche heuristiques bien plus efficaces, bien que l'efficacité d'un algorithme est toujours dépendante du domaine fixé : certains algorithmes se prêtent mieux à certains type de domaines, et les algorithmes les plus généraux ont tendance à avoir une performance moins bonne que des algorithmes spécialisés.

Enfin, il est intéressant de noter qu'il est possible d'utiliser l'heuristique au sein de techniques

plus complexes : nous avons évoqué que dans des algorithmes tels que A* l'heuristique est utilisée en remplacement d'un coût qui n'est pas connu ; cependant, il est possible de favoriser l'heuristique en établissant le coût d'une action comme étant le coût immédiat (connu) ajouté à $W * h(s)$ où $h(s)$ est l'heuristique de l'état d'arrivée de l'action et $W \geq 1$ est une constante. Avec de telles méthodes, il est possible d'obtenir une solution plus rapidement, puisqu'on favorise la connaissance apportée par l'heuristique, mais les solutions obtenues ont bien souvent une moins bonne qualité.

En nous basant sur les concepts que nous avons détaillés dans les paragraphes précédents, nous allons à présent formaliser un algorithme de résolution pour le problème PCMDP qui utilise ces techniques pour explorer l'espace de recherche de façon efficace. Bien que nous n'utiliserons que les concepts basiques des heuristiques, tels que la régression par omission des effets de suppression, le cadre que nous présentons se prête bien à l'utilisation d'autres techniques plus avancées ; nous nous limiterons cependant à des heuristiques assez simples, afin de pouvoir prouver certaines propriétés de l'algorithme telles que la complétude ou l'optimalité.

VII.2 Développement d'un algorithme de résolution pour PCMDP basé sur des techniques de recherche heuristique

Nous détaillons dans les paragraphes suivants un algorithme permettant de trouver la politique déterministe valide et optimale pour un PCMDP, lorsqu'elle existe.

Comme nous l'avons évoqué précédemment, cet algorithme est inspiré des algorithmes de type Branch and Bound. Ici, nous effectuons le branchement sur l'espace des politiques partielles : à chaque étape de l'algorithme (Algorithme 12 page 156), nous prenons une politique partielle, pour laquelle nous étendons un certain nombre de politiques partielles filles ; ces politiques partielles filles sont obtenues en fixant une des actions non fixée dans un des états non-étendus ; de façon plus précise, nous choisissons un état s pour lequel nous étendons toutes les actions possibles, ce qui nous donne autant de politiques partielles possibles (Figure 28 page 155).

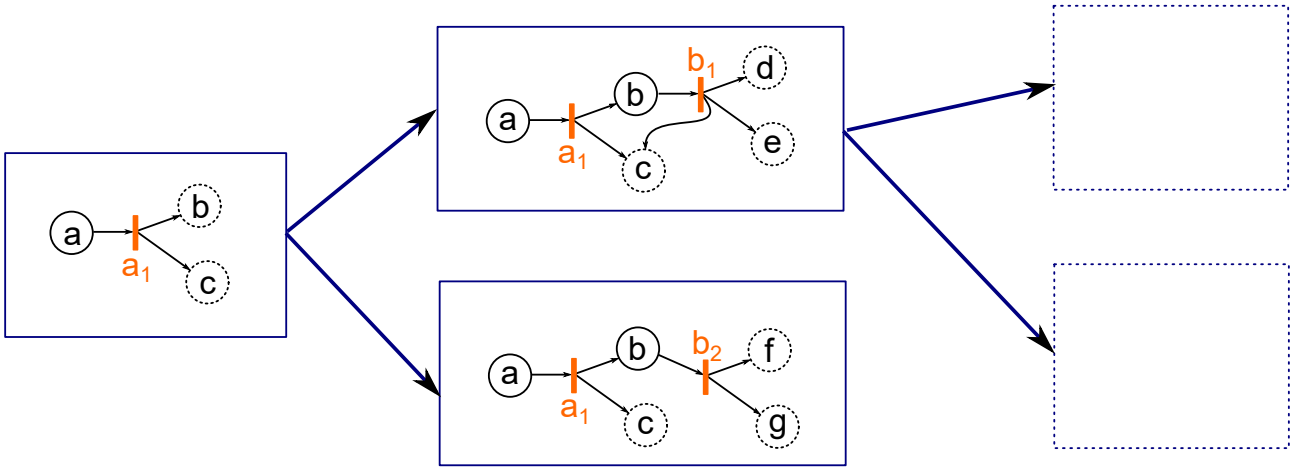


FIGURE 28 – Branch and Bound : on construit un arbre où les politiques partielles filles sont obtenues en étendant toutes les actions d'un des états non-étendus de la politique partielle mère

Pour chacune des politiques partielles étendues de cette façon, nous pouvons dans un premier temps déterminer si elle est valide : si nous savons que cette politique partielle viole au moins une contrainte, alors nous pouvons la supprimer immédiatement de la recherche. Ceci est effectué dans la fonction isValid (Algorithme 13 page 157) : cette fonction est simplement le calcul de la validité pour chaque contrainte à l'aide de l'algorithme de mise à jour itératif pour PCTL (Théorème 2 page 22). Puisque certains états de la politique partielle ne sont pas étendus, la valeur de ces états est obtenue à partir d'une heuristique d'atteignabilité ; dès lors qu'il est établi qu'une contrainte est violée, alors il est possible de supprimer la politique partielle de l'arbre de recherche ; sinon, cette politique est ajoutée à l'ensemble des politiques à étendre dans la suite de l'algorithme de Branch and Bound.

Puis, si la politique fille est valide, nous regardons si elle est complète, c'est-à-dire si tous les états qui peuvent être atteints avec cette politique ont été associés à une action ; si c'est le cas, alors il s'agit d'une politique solution : on compare cette politique (complète) avec la meilleure politique complète qui a été trouvée jusqu'ici, en gardant la meilleure des deux en termes de valeur (au sens de la récompense cumulée espérée). Nous pouvons par ailleurs supprimer de l'exploration toutes les politiques partielles qui ont une valeur plus faible que la nouvelle solution trouvée. Si la politique n'est pas complète, alors elle est ajoutée à l'ensemble des politiques à étendre.

Le dernier aspect de l'algorithme concerne l'ordre de sélection des politiques dans l'ensemble des politiques à étendre. L'ensemble des politiques à étendre est en réalité une file de priorité, dans laquelle les politiques sont classées à chaque instant ; nous détaillerons le critère de choix de cette politique ultérieurement, mais nous pouvons mentionner qu'il prend en compte la valeur d'une politique partielle ; cette valeur est calculée par une simple propagation de la valeur pour tous les états visités par la politique partielle, ce qui est effectué dans la fonction computeValue (Algorithme 14 page 158). À nouveau, comme tous les états d'une politique partielle ne sont pas étendus, nous avons recours à une heuristique pour estimer la valeur de ces états.

VII.2.1 Formalisation de l'algorithme de résolution

L'algorithme complet est défini dans l'algorithme ci-après (Algorithme 12 page 156). Comme nous l'avons détaillé dans les sections précédentes, la première étape est, au besoin, de transformer les contraintes non-transitoires et les contraintes à horizon fini en des contraintes qui soient transitoires et à horizon infini. Ceci peut être effectué par l'ajout de variables de mémoire dans l'espace d'états, suivi de la modification des fonctions f et g à la volée.

Algorithm 12: FastPCMDP : algorithme de Branch-and-Bound dans le cadre des PCMDP

```
1  Procédure FastPCMDP (S, A, R, T, s_0, γ, ξ);
   Entrées  : ⟨S, A, R, T, s_0, γ, ξ⟩ tels que définissants un PCMDP avec contraintes ξ_1, ..., ξ_n
   Sorties  : π politique déterministe optimale
   Variables: Tip l'ensemble des politiques partielles
              p_0 la politique partielle où seul s_0 est étendue
              BestP la meilleure solution trouvée
2  ---------------------------------------------------------------------
3  Convertir les contraintes transitoires et les horizons si nécessaire ;
4  Ajouter p_0 à Tip;
5  while Tip n'est pas vide do
6      Sélectionner une politique P ∈ Tip;
7      Sélectionner un état s ∈ P.U, états non étendus de P ;
8      for Action a ∈ A do
9          Créer une politique partielle P^(a) où l'action a est appliquée en s ;
10         Mettre à jour les états non-étendus de P^(a) ;
11         Calculer la valeur : P^(a).value = updateValue(P^(a));
12         if isValid(P^(a)) then
13             if P^(a).U est non vide then
14                 Ajouter P^(a) à Tip;
15             else if P^(a).value > BestP.value then
16                 BestP=P^(a) ;
17 Retourner BestP;
```

VII.2.2 Preuves de complétude de l'algorithme

Comme nous l'avons marqué, l'algorithme est garanti de trouver une solution, lorsqu'une solution existe, tant que la fonction d'évaluation présente dans $isValid()$ est admissible : $isValid()$ doit retourner $Vrai$ dès lors qu'il existe une politique valide complète ; cette fonction a en revanche le droit de retourner $Vrai$ même s'il n'existe plus de politique valide fille. Ceci signifie que dans $isValid()$ toutes les heuristiques utilisées doivent être admissibles.

Pour améliorer la performance de l'algorithme, il est possible de supprimer à chaque instant toutes les politiques partielles de Tip (l'ensemble de toutes les politiques partielles explorées) qui ont une valeur inférieure ou égale à la meilleure politique déjà trouvée $BestP$. Ceci peut être effectué en vérifiant si $P.value > BestP.value$ au moment de la sélection d'une nouvelle politique partielle $P \in Tip$. Cependant, pour garantir que l'algorithme soit complet - c'est-à-dire qu'il trouve toujours une solution lorsqu'il en existe une - l'évaluation du coût (et l'heuristique de coût) doivent être admissibles : une fonction heuristique donnant la récompense attendue d'un état (ou d'une action non étendue) doit être supérieure ou égale à la valeur réelle de l'espérance de la récompense. Si ce n'est pas le cas, il y a potentiellement un risque que certaines solutions se retrouvent supprimées à tort par l'algorithme.

En gardant à l'esprit ces considérations, les deux procédures sont relativement classiques : il s'agit de propager des valeurs au travers d'itérations de l'opérateur de Bellman, en démarrant respectivement

Algorithm 13: isValid(P)

```
1  Fonction isValid (P);
   Entrées   : P : S → A une politique partielle
   Variables : h : S → ℝ une fonction heuristique sur les états
   Sorties   : Vrai si la politique partielle a potentiellement une politique complète fille valide
2  ──────────────────────────────────────────────────────────────────────
3  for chaque contrainte ξ do
4      Fixer ∀s ∈ S, W(s) = g_ξ(s), la probabilité de valider la contrainte dans un état;
5      while certaines probabilités W(s) ont changé do
6          for s ∈ S do
7              if s n'est pas étendu then
8                  W(s) = h(s);
9              else
10                 Soit a = P(s) l'action sélectionnée en s;
11                 W(s) = Σ_{s'∈S} T(s, a, s')W(s')
12     if !(W(s_0) >_ξ p_ξ) then Retourner Faux;
13 Retourner Vrai;
```

depuis les états où g est vraie vers l'état initial, et depuis les états ayant des récompenses vers l'état initial. La valeur (de validité ou de récompense) d'un état non exploré est donnée par une heuristique d'estimation. Puisqu'il s'agit d'une simple propagation, chaque passage de ces deux fonctions est donc très peu coûteux en termes de temps.

Notons que nous supposons ici, par souci de lisibilité, que les contraintes sont transitoires ; en particulier, nous faisons l'hypothèse que pour chaque contrainte, lorsqu'un chemin atteint un état où la fonction f est fausse, alors il ne peut jamais atteindre un état où g est vraie. Ceci explique pourquoi la fonction f n'apparaît jamais explicitement dans l'algorithme. Il est trivial d'adapter l'algorithme $isValid()$ pour explicitement arrêter la propagation lorsque f est fausse, de la même manière que ce que nous avons pu faire pour l'algorithme SPC MDP [SKTK14].

Preuve de terminaison

L'algorithme explore l'espace des politiques partielles ; puisqu'il y a un nombre fini d'états et d'actions, il y a un nombre fini de politiques partielles.

On peut facilement prouver qu'une politique partielle donnée n'est jamais ajoutée deux fois dans l'ensemble Tip : puisque deux politiques partielles données dans Tip doivent avoir un ancêtre commun, nous pouvons prendre le plus récent de ces ancêtres communs, qui est donc une politique telle que le prochain état étendu ait une action différente dans les deux listes des ancêtres de ces politiques.

Par conséquent, le nombre de politiques partielles visitées (qui ont été ajoutées une fois dans Tip) est un nombre croissant strictement croissant avec chaque boucle, avec une borne maximale. Ceci prouve la terminaison de la boucle principale, et donc de l'algorithme.

Preuve de validité

Si une solution a été trouvée, elle a à un certain moment été testée par la fonction $isValid()$. Puisqu'une solution ne doit avoir aucun état non-étendu, tous les états de cette solution sont étendus, donc $isValid()$ est réduit au même algorithme que celui permettant de calculer la validité d'une politique ([HJ94] et (Théorème 2 page 22)). Ceci prouve que la solution obtenue respecte toutes les contraintes.

Algorithm 14: computeValue(P)

```
1  Fonction computeValue (P);
   Entrées   : P : S → A une politique partielle
   Variables : h_r : S → ℝ une fonction heuristique sur les états
   Sorties   : V(s_0) la valeur de la politique partielle
2  ──────────────────────────────────────────────────────────
3  Fixer ∀s ∈ S, V(s) = 0, la valeur d'un état;
4  while certaines valeurs V(s) ont changé do
5      for s ∈ S do
6          if s n'est pas étendue then
7              V(s) = h_r(s);
8          else
9              Soit a = P(s) l'action sélectionnée en s;
10             V(s) = R(s,a) + γ Σ_{s'∈S} T(s,a,s')V(s')
11 Retourner V(s_0);
```

Preuve d'optimalité

À chaque fois qu'une solution est sélectionnée comme étant la meilleure, elle a été comparée avec la meilleure solution trouvée pour l'instant, ce qui signifie qu'elle a la valeur la plus élevée parmi toutes les politiques complètes (sans états non-étendus) valides précédemment trouvées. Cette valeur est la *valeur réelle*, puisque la politique partielle est seulement considérée comme étant une solution à partir du moment où elle est complète (elle est entièrement étendue), c'est-à-dire par conséquent lorsque la fonction $computeValue()$ est réduite à la fonction de calcul de la Valeur classique. Cette valeur ne dépend pas de l'heuristique.

Pour prouver l'optimalité, il nous faut alors seulement garantir que toutes les politiques valides sont comparées, à un certain moment, avec la meilleure politique courante. Ceci est vrai si au moins une des deux conditions suivantes est vérifiée : (1) la fonction d'heuristique dans $updateValue(P)$ est admissible, ce qui signifie qu'elle surestime toujours la valeur de la politique partielle, ou (2) on ne supprime jamais de politique partielle en se basant sur sa valeur, on ne lui affecte qu'une priorité plus faible - contrairement à ce que nous avons pu proposer de faire précédemment. Le choix d'imposer l'une ou l'autre de ces conditions repose sur l'utilisateur de l'algorithme, même si nous détaillerons dans une section ultérieure la condition que nous avons choisie.

VII.2.3 Sélection des paramètres de l'algorithme

En ayant défini ces fonctions d'évaluations, quatre points restent à éclaircir :
- Comment s'effectue le choix de la politique partielle $P \in Tip$ à étendre ?
- Quel état non-étendu s de P faut-il choisir d'étendre ?
- Comment pouvons nous trouver les fonctions heuristiques h et h_r ?
- À quel moment doit-on décider d'arrêter les boucles de mise à jour dans $computeValue()$ et $isValid(P)$?

Les réponses à ces questions n'ont pas été écrites dans l'explication de l'algorithme, puisqu'elles n'impactent ni la validité ni l'optimalité : il serait par exemple possible de choisir un ordre aléatoire, et l'algorithme serait toujours correct, pour peu que l'heuristique d'atteignabilité soit admissible. Cependant, les réponses à ces questions permettent vraisemblablement d'améliorer de façon significative le temps de calcul.

Sélection des stratégies de priorétisation

L'ordre de sélection des politiques partielles à étendre a une incidence importante sur la forme des politiques solutions : si deux politiques solutions existent - c'est-à-dire ayant la même valeur et étant

toutes deux valides - l'une passant par un état s_1 tandis que l'autre préfère une autre action passant par un état s_2, alors notre algorithme retournera comme solution la première des deux politiques trouvées lors de l'exploration. Ainsi, en spécifiant les critères d'ordre pour la sélection des politiques partielles, nous spécifions des préférences sur la forme des solutions.

Nous avons appliqué les priorités suivantes lors du tri de Tip :

1. Les politiques partielles avec la **plus grande valeur** sont choisies en premier.
2. Lorsque les valeurs sont égales, on calcule la **distance au but** comme défini dans la définition (Définition 46 page 159). Les politiques partielles avec les plus faibles distances au but sont choisies en premier. Ceci favorise les politiques partielles qui remplissent tous les objectifs rapidement après avoir collecté toutes les récompenses.
3. Si les distances sont égales, les politiques partielles avec le **plus petit nombre d'états non-étendus** sont choisies en premier. Ceci favorise les politiques partielles qui font en sorte que plusieurs chemins se rejoignent, plutôt que celle explorant des chemins inconnus.
4. Si toutes ces conditions sont égales, les politiques partielles avec le **plus petit nombre d'actions fixés** sont choisies en premier. Ceci correspond à une préférence pour l'exploration en largeur d'abord (Breadth-First).

Définition 46 (*Distance au but*)

Soit P une politique partielle, $P.U$ son ensemble d'états non-étendus.
Soit $\mathcal{S}^h$ un espace d'états heuristique, c'est-à-dire un espace constitué d'une version relâchée de chaque état, associée aux actions et transitions appropriées.
Pour $s \in S$, on définit pour chaque contrainte ξ la distance heuristique $d_\xi(s, G_\xi)$ comme étant le nombre minimal d'actions dans l'espace d'états heuristique qui permet d'atteindre un état où g_ξ est vraie depuis s.
On définit pour chaque contrainte ξ la distance moyenne au but pour cette contrainte comme étant :

$$d_\xi(P, G_\xi) = \frac{\sum_{s \in P.U}(d_\xi(s, G_\xi))}{\#P.U}$$

où $\#P.U$ est le nombre d'états dans $P.U$. Enfin, on définit la distance au but comme (n étant le nombre de contraintes) :

$$d(P) = \frac{\sum_{i=0}^{n} d_{\xi_i}(P, G_{\xi_i})}{n}$$

Cet ordre de sélection nécessite d'avoir un tel niveau de détail puisque, lorsque les fonctions heuristiques ne sont pas très informatives, nous rencontrerons beaucoup de cas d'égalité lors de l'exploration et il nous faut être capable de favoriser un chemin par rapport à un autre. Avec ces critères, on formalise la forme des solutions que nous cherchons : nous cherchons celles qui ont la meilleure valeur, puis celles qui atteignent les buts le plus rapidement, sans explorer plus que nécessaire et globalement avec le nombre minimum d'actions.

Sélection d'un état à étendre

En fin de compte, tous les états non-étendus seront étendus à un certain moment ; l'ordre de sélection n'est donc pas critique, puisqu'il ne donne pas immédiatement de moyen de couper certaines politiques partielles. Cependant, nous pouvons suivre deux stratégies principales pour améliorer la vitesse de notre algorithme :

- **(Option 1)** étendre l'état non-étendu le plus visité. Ceci permet de voir rapidement si une contrainte sera invalide ou non. Toutefois, cette évaluation vient avec un certain coût de calcul pour déterminer pour chaque politique partielle quel est l'état non-étendu le plus visité.
- **(Option 2)** étendre l'état non-étendu qui a la plus petite *distance à un but*. Ceci permet de voir rapidement si l'heuristique d'atteignabilité s'est ou non trompée, mais peut ne pas donner suffisamment d'information pour arrêter l'exploration. Le calcul est cependant immédiat, puisqu'il est inclus dans le calcul de l'heuristique.

Pour notre solveur, nous avons implémenté l'(Option 2), puisque le défaut principal des cas de tests était que l'heuristique d'atteignabilité était peu informative jusqu'au moment où l'on approche d'un état but. Cependant, l'(Option 1) est de façon intuitive la stratégie la plus efficace dès lors que les contraintes sont dures à remplir, et lorsqu'on espère que nous pourrons utiliser cette information pour arrêter la recherche au plus tôt.

Calcul des fonctions heuristiques

Nous avons besoin de deux types de fonctions heuristiques : une donnant pour chaque état non-étendu une évaluation de la probabilité maximale d'atteindre (ou d'éviter) un état but g, et l'autre donnant pour chaque état non-étendu une évaluation de l'espérance de la récompense dévaluée maximale.

Le calcul de ces fonctions peut être effectué de la même manière pour les deux heuristiques : nous pouvons résoudre une version simplifiée de chaque problème et utiliser le résultat comme une fonction d'heuristique. La simplification classique consiste à supprimer toutes les probabilités des effets et de supprimer les effets *DEL* lorsque le problème est exprimé sous la forme de fluents ADD/DEL. Notons que nous avons choisi le formalisme en fluent booléens au travers du langage PPDDL [YS04], qui conditionne la structure de cette heuristique : pour chaque domaine, il existe des heuristiques spécifiques qui sont plus efficaces que l'heuristique par défaut que nous proposons.

Avec plus de détails : à la création de chaque plan partiel nous créons un nouvel objet, qui contient tous les états et toutes les actions du plan partiel, ainsi que les états terminaux laissés ouverts. L'algorithme de calcul de l'heuristique va étendre tous les états terminaux récursivement, jusqu'à ce que tous les états aient été étendus. Cette extension se fait dans l'espace simplifié - aussi appelé espace relâché - c'est-à-dire un espace où tous les effets DEL sont enlevés, tous les effets probabilistes ADD sont combinés pour ne donner qu'une seule transition comportant tous les effets ADD, et les récompenses des effets probabilistes deviennent le maximum des récompenses de toutes les issues possibles. L'algorithme va alors effectuer une propagation simple pour calculer la valeur de chaque état du plan partiel ; la valeur de l'heuristique d'atteignabilité ou de récompense des états non étendus de la politique partielle d'origine sont alors directement les valeurs obtenues par la procédure de Value-Iteration dans l'espace d'états relâché. D'une certaine manière, l'heuristique correspond à une recherche en avant dans l'espace relâché, aussi appelé **"lookahead"** dans la littérature anglophone.

Notons qu'il aurait été possible de choisir une heuristique moins informative mais plus rapide, par exemple en s'inspirant des heuristiques h_{add} ou h_{max} qui n'explorent pas entièrement l'espace des états relâchés, mais calculent la difficulté d'établissement de chaque fluent par une exploration vers l'avant. Cependant, il est essentiel de noter que l'adaptation de ces heuristiques dans le cadre des MDP n'est pas triviale, puisque les MDP présentent plusieurs caractéristiques que n'ont pas les problèmes de planification classique, telles que la possibilité que les politiques optimales atteignent des états impasses avec une certaine probabilité, ou encore la prise en compte de la dévaluation dans le calcul de la valeur. Des travaux réalisant des adaptations de telles heuristiques existent [TKVI11], mais une étude spécifique à chaque heuristique demeure nécessaire pour étudier dans quelle mesure elle s'applique réellement dans notre cas, et conserve des propriétés intéressantes. De façon générale, cependant, nous pouvons nous attendre à ce que toute fonction heuristique permettant d'obtenir de bonnes performances pour une résolution par un algorithme de type LAO* pourra être convertie immédiatement en heuristique pour notre algorithme. Notons qu'il pourra s'agir d'heuristiques différentes pour l'atteignabilité ou pour la valeur.

Ces simplifications donnent effectivement des heuristiques admissibles pour la fonction d'atteignabilité, puisqu'elles affirment qu'un but peut être atteint à chaque fois qu'il peut effectivement être atteint ; ceci peut-être prouvé en suivant une démonstration similaire à celle de [TKVI11], ou de travaux similaires. Pour l'heuristique de récompense, ces simplifications ne sont pas garanties comme étant admissibles : elles peuvent affirmer un coût plus faible qu'en réalité, en particulier si on utilise un opérateur + au lieu de l'opérateur *max* pour la fusion des effets probabilistes. Afin de garantir que l'algorithme est complet, il nous faudrait alors garder tous les plans partiels dans Tip, malgré le fait que certains aient une valeur estimée nous permettant de les supprimer. Toutefois, puisque nous devons explorer un nombre immense de politiques partielles, ces politiques partielles seront vraisemblablement

explorées en dernier ; dans nos cas de test, nous avons par conséquent choisi de les supprimer.

Enfin, notons que nous aurions pu choisir de ne réaliser le calcul de l'heuristique qu'à l'état initial, plutôt que pour chaque état étendu ; néanmoins, nos expérimentations ont montré que sans mise à jour, l'heuristique n'avait pas un niveau d'information suffisant pour permettre des performances intéressantes. Au final, notre approche s'est avérée mixte : pour chaque nouvelle politique partielle, nous créons un objet de calcul de l'heuristique à partir de la politique partielle mère, en effectuant les connexions manquantes vers l'espace des états relâchés. De cette manière, il n'est nécessaire d'effectuer qu'une faible ré-exploration de l'espace des états relâchés.

Condition de terminaison des boucles de mise à jour.

Pour les fonctions $isValid()$ et $computeValue()$, il est nécessaire de répondre à la question de la condition de terminaison de la boucle *while* : dans l'algorithme classique de mise à jour de Bellman, on définit un paramètre ϵ auquel on compare à chaque étape de la boucle l'écart entre les anciennes et les nouvelles valeurs. Cependant, calculer ϵ par avance peut s'avérer difficile : si l'on ne choisit pas une valeur assez faible pour ϵ, il est possible que la fonction s'arrête avant que l'état initial ait reçu suffisamment de valeur de récompense, ce qui peut amener l'algorithme de Branch-and-Bound à arrêter immédiatement l'exploration de cette branche. À l'opposé, si on choisit une valeur trop faible pour ϵ, il y aura beaucoup trop de calculs effectués pour une branche qui peut s'avérer comme étant au final une impasse.

Il est possible de trouver deux solutions pour arrêter le calcul au plus tôt : (1) dans $isValid()$, nous pouvons tester à la fin de chaque boucle si le seuil de probabilité a déjà été validé, au lieu d'attendre la fin de la fonction ; et (2) nous pouvons fixer un compteur sur le nombre d'itérations, qui causera la fonction à retourner un signal de Time-Out ; alors, pour assurer que l'algorithme demeure complet, nous pouvons garder les politiques partielles où la fonction est arrivé à un Time-Out, mais en les mettant avec une priorité plus faible que les politiques partielles que nous savons être valides. De façon expérimentale, un compteur de Time-Out fixé à 2 fois le nombre d'états explorés semble atteindre un équilibre satisfaisant, nous permettant de fixer ϵ à 0. À nouveau, ce paramètre n'impacte ni la validité ni l'optimalité de l'algorithme, mais impactera de façon significative le temps de calcul, de façon positive comme de façon négative.

Notons par ailleurs qu'il est possible de trouver un ordre optimal pour la propagation des valeurs dans ces deux fonctions, par exemple en propageant les valeurs depuis les états les plus éloignés de l'état initial. Néanmoins, nos expérimentations ont montré que de telles améliorations apportaient un gain marginal sur nos cas d'application, principalement puisque la qualité des heuristiques s'est avérée être le facteur limitant ; pour d'autres domaines, il est en revanche possible que ce type d'amélioration représente un gain mesurable, en particulier sur des domaines où les récompenses se trouvent sur des états éloignés par rapport à l'état initial. Pour ce type de domaine, il est par ailleurs intéressant de maintenir des variables supplémentaires dans les politiques partielles, pour par exemple ne mettre à jour les valeurs de récompense ou d'atteignabilité que lorsque des actions à récompense non-nulle ou des états buts "g" ont été atteints lors de l'expansion.

VII.3 Évaluation des performances de l'algorithme Fast-PCMDP sur un ensemble de cas de tests

Pour évaluer les performances de notre algorithme, nous avons réalisé un ensemble de tests sur des exemples classiques, avec l'ajout de contraintes PCTL. Les expériences ont été menées sur un ordinateur portable ayant un processeur 2*2.40 GHz et 8Go de RAM.

Tous les tests, exceptés le domaine Gridworld, ont été réalisés sur l'extension du langage PPDDL [YS04] que nous avons déjà évoquée précédemment, qui comporte l'ajout du mot clé *(PCTL f g > p)* pour exprimer des contraintes PCTL.

L'objectif de cette évaluation est triple : (1) nous souhaitons évaluer la performance en temps de calcul de notre algorithme vis-à-vis des autres algorithmes existants de résolution PCMDP et SPC MDP ; (2) nous souhaitons évaluer si l'algorithme est plus performant sur un spectre particulier de paramètres, par exemple en fonction du nombre de contraintes ou de la probabilité des contraintes ; et (3) nous souhaitons évaluer dans quelle mesure des heuristiques informatives peuvent être trouvées dans des cas pratiques.

En effet, notre algorithme repose en grande partie sur l'exploitation de la connaissance apportée par les deux heuristiques : en ayant des heuristiques parfaites, la résolution serait immédiate, tandis qu'en ayant des heuristiques admissibles mais peu informatives elle correspondrait à une résolution par force brute. En effectuant des tests, nous pouvons ainsi avoir un aperçu de la précision requise pour ces heuristiques et de la difficulté d'obtenir cette précision sur des cas réels.

VII.3.1 Comparaison de la performance en temps de calcul vis-à-vis de l'algorithme PCMDP-ILP

Le cas de test nous permettant de comparer au mieux les performances de notre algorithme avec les autres algorithmes PCMDP est celui du **Gridworld** qui a été décrit initialement dans l'article de Teichteil [TK12a] définissant le cadre PCMDP. Cette comparaison a été possible puisque nous avons pu disposer du solveur PCMDP-ILP décrit dans ce même article, ce qui n'a pas été possible avec d'autres solveurs qui auraient pu potentiellement servir de point de comparaison (Figure 9 page 42) ; cependant, les résultats mis en avant dans les autres articles de la littérature semblent au premier abord présenter des temps de calcul similaires à PCMDP-ILP.

Le domaine du Gridworld est constitué d'une grille $n \times n$. L'agent choisit d'aller dans une des quatre directions (Haut, Bas, Gauche, Droite) et le résultat de chaque action a une probabilité faible d'aller de côté : l'agent se rend dans la direction attendue avec une probabilité 0.8, mais se rend à la cellule à droite ou à gauche de la direction avec pour chacune une probabilité 0.1.

Certaines cellules de cette grille sont marquées avec une récompense $+1$ ou -1 utilisée une seule fois : l'agent gagne une récompense $+1$ ou -1 lorsqu'il entre dans cette cellule ; une variable mémoire dans l'espace d'états permet de se rappeler que cette récompense a été collectée.

D'autres cellules ont des contraintes exprimées en PCTL : certaines zones ont des contraintes d'atteignabilité $(true)U^{>p}g$, où les zones marquées avec g doivent être atteintes avec une probabilité d'au moins p ; d'autres ont des contraintes d'évitement $(true)U^{<p}g$, où les zones marquées avec g doivent être évitées - elles doivent être atteintes avec au plus une probabilité p. Toutes les zones marquées par une fonction g sont absorbantes. Ceci assure de plus que toutes les contraintes sont transitoires.

Le but de l'agent est d'atteindre certains des objectifs avec une probabilité suffisante (contraintes d'atteignabilité), tout en évitant les pièges (contraintes d'évitement) et en collectant autant de récompenses que possible le long du chemin.

Un exemple d'instance générée est représenté sur la figure 29 : une cellule orange représente l'état initial, une cellule verte une récompense unique $+1$, rouge pour une pénalité -1, bleue pour une contrainte PCTL d'atteignabilité et gris pour une contrainte PCTL d'évitement.

Ce domaine a été choisi spécifiquement puisqu'il permet une comparaison avec l'algorithme PCMDP-ILP [TK12a]. Comme nous pouvons le voir dans le tableau (Tableau 3 page 163), notre algorithme est plus rapide de plusieurs ordres de grandeur que l'algorithme PCMDP-ILP, basé sur une approche par

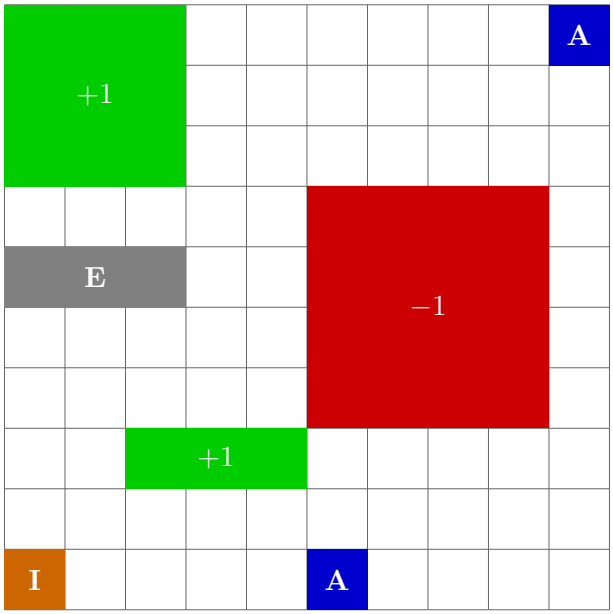


FIGURE 29 – Exemple d'instance du problème Gridworld

programmation linéaire. Sur le tableau, les valeurs correspondent aux temps moyens de résolution de plusieurs instances de ce problème, générés aléatoirements avec différentes zones.

n	Fast-PCMDP (s)	PCMDP-ILP (s)
10	0.0446754	0.004231
25	0.222472	0.15949
40	0.657353	1.82726
50	0.819725	4.93267
60	2.35722	129.893
75	2.0088	408.694
100	8.74216	> 3600
150	9.87435	
200	19.5581	
300	107.765	

TABLE 3 – Comparaison des temps sur le domaine Gridworld

L'une des hypothèses qu'il est important de noter est celle portant sur le fait que les récompenses ne sont collectées qu'une seule fois. Ceci rend le domaine plus adapté à notre algorithme, puisque cela réduit les conflits entre les stratégies cherchant un chemin avec le plus de récompenses et les stratégies cherchant à remplir les différents objectifs. Si les récompenses pouvaient être collectées à l'infini, la valeur de la meilleure politique solution trouvée par notre algorithme aurait été bien plus basse que la valeur d'une vraie politique optimale, qui aurait alors été une politique aléatoire.

Cependant, il est intéressant de noter que le choix des politiques déterministes conduit la plupart du temps à des trajectoires plus intuitives que les trajectoires trouvées avec une politique aléatoire, indépendamment de si la récompense est ou non finie : avec des récompenses collectées à l'infini, la trajectoire d'une politique aléatoire cherchera toujours à collecter les récompenses, avec seulement une faible probabilité de choisir un chemin différent vers les objectifs. Lors d'une simulation, ceci donne donc l'impression que l'agent tombe par hasard sur le but à un certain moment, plutôt que d'y aller de façon intentionné.

À l'inverse, nous pouvons voir dans le petit exemple (Figure 30 page 164) que notre approche mène à des trajectoires claires et intuitives : l'agent se dirige en premier lieu vers les récompenses, puis directement vers les objectifs. Ceci est dû à l'ordre de priorité que nous avons fixé lors du tri

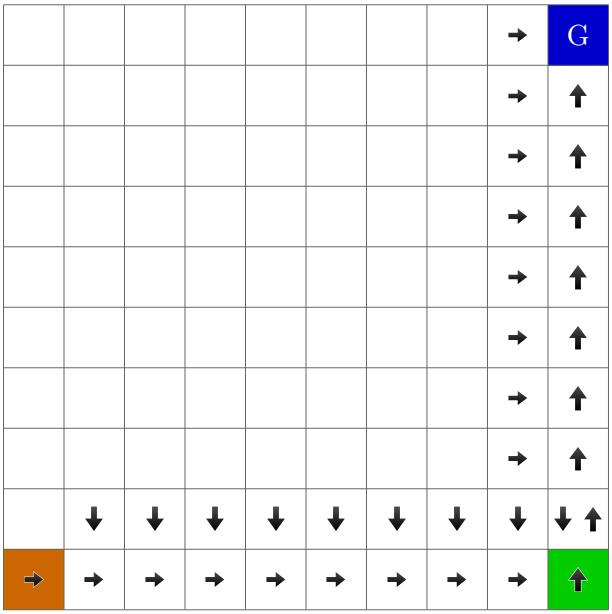


FIGURE 30 – Exemple d'une solution pour un problème simple de Gridworld

de l'ensemble des politiques partielles dans l'algorithme : nous avons choisi de trier en priorité sur la valeur, puis sur la distance au but, puis sur le nombre d'états ouverts (i.e. la dispersion de l'algorithme). Même avec une récompense unique, une politique optimale aléatoire ne va pas nécessairement droit vers un objectif après avoir collecté toutes les récompenses, si le problème n'impose pas ce comportement avec des récompenses positives lors de la complétion des objectifs.

Notons que la trajectoire sur la figure (Figure 30 page 164) a été simplifiée : le système a en vérité une variable enregistrant si la récompense a ou non été collectée ; dans la cellule avec deux flèches, l'agent ne descend que si la récompense n'a pas été collectée.

VII.3.2 Évaluation sur des cas de tests académiques

À partir de l'évaluation précédente, nous savons donc que notre algorithme est sensiblement plus rapide que PCMDP-ILP sur le domaine Gridworld, ce qui nous permet d'espérer raisonnablement qu'il puisse être plus rapide sur d'autres domaines. Cependant, le domaine Gridworld présente l'avantage principal de permettre la création d'heuristiques très intuitives, sous la forme de la distance entre les coordonnées X et Y de la position actuelle de l'agent et de chacun des objectifs. Cette heuristique n'est pas parfaite, puisque le chemin le plus court peut nécessiter de faire un détour pour éviter une zone interdite, mais elle est suffisamment informative pour restreindre considérablement l'espace de recherche : vraisemblablement, si tous les objectifs sont au Nord alors l'agent devra privilégier la recherche des politiques se dirigeant vers le Nord, ce qui élimine d'une certaine façon trois-quart des options pour chaque cellule visitée.

Nous souhaitons à présent évaluer comment l'algorithme se comporte sur un domaine où une heuristique intuitive n'existe pas. L'un des domaines classiques de la littérature les plus difficiles est celui du **Blocksworld**, que nous présentons dans les paragraphes suivants.

Le domaine Blocksworld est un domaine classique, en particulier puisque sa description au format PPDDL est disponible dans le cadre de la compétition de planification IPC (International Planning Competition) [ICA]. Il consiste en n blocs, qui doivent être déplacés les uns au dessus des autres. Chaque action a une probabilité d'échouer, c'est-à-dire que le cube qu'on tente de déplacer est lâché sur la table par inadvertance. Le but est d'assembler les blocs en une tour selon un agencement spécifique.

Ce domaine est rendu difficile en particulier parce que tous les états peuvent être atteints les uns à

Nombres de cubes	Fast-PCMDP (s)	% de complétion
5	0.02553085	100
10	9.98608	30
15	>3600	0

TABLE 4 – Résultats pour le domaine Blocksworld

partir des autres. Notre algorithme n'est pas purement dédié à la recherche de but, il ne s'est ainsi pas avéré capable de résoudre des instances non-triviales sans aide supplémentaire. Cependant, nous avons obtenu des résultats satisfaisants avec l'ajout d'une seule contrainte PCTL, disant que le système devait à un certain moment avoir tous les blocs sur la table au même instant, avec une forte probabilité. Ceci nous a permis de guider le système pour qu'il se rende dans un premier temps vers ce but intermédiaire, avant de compléter l'objectif consistant à faire une tour particulière, qui était exprimé sous la forme d'une récompense. Le résultat peut être vu au travers du tableau (Tableau 4 page 165), dans lequel les temps sont moyennés sur plusieurs instances et qui présente aussi le nombre d'instances complétées : lorsque certaines instances étaient trop complexes, l'algorithme ne trouvait pas le but à temps (3 600s de time-out) ou utilisait toute la mémoire disponible.

Dans une certaine mesure, la contrainte que nous avons ici utilisée agit comme une heuristique qualitative, sous la forme d'un but intermédiaire, de manière similaire à ce qui peut être fait pour les landmarks en planification [HPL04].

Les résultats se sont avérés similaires avec le domaine Exploding Blocksworld, où des explosions peuvent se produire de façon aléatoire à chaque action.

Un lecteur intéressé pourra se référer à de nombreux articles pour comparer ces résultats avec des algorithmes de recherche heuristique existants [GNT04]. La conclusion que nous pouvons en tirer est que l'heuristique par défaut que nous utilisons est bien moins efficace en termes de temps de calcul que les méthodes utilisées par les algorithmes dédiés à la recherche ; en effet, puisque l'heuristique par défaut repose sur l'exploration de la totalité de l'espace relâché, elle n'est pas adaptée à des domaines où l'espace relâché présente trop de fluents différents. Cependant, même avec une heuristique plus adaptée, notre algorithme de résolution aurait toujours une perte de performance due à la sur-couche de Branch-and-Bound, ce qui le rend dans tous les cas moins efficace qu'un algorithme plus direct tel qu'un algorithme de Greedy-Best-First Search.

VII.3.3 Évaluation sur le problème du Business Jet

Les paragraphes précédents nous ont permis d'évaluer la performance de notre algorithme face à des heuristiques précises ou imprécises. Les conclusions préliminaires que nous pouvons en tirer sont que le niveau d'information apporté par les heuristiques est capital pour la performance de l'algorithme, en particulier puisque le nombre de branches explorées impacte à la fois le temps de calcul et la mémoire requise pour la résolution.

Le second paramètre majeur que nous pouvons faire varier est celui de la difficulté d'accomplissement des contraintes : de façon intuitive, une contrainte d'atteignabilité avec une probabilité $p > 0.5$ semble plus facile à respecter qu'une contrainte d'atteignabilité avec une probabilité $p = 1$; cependant il n'est pas aisé de voir dans quelle mesure cela impactera le temps de calcul, puisqu'une contrainte difficile permettra de couper plus de branches lors de l'exploration tandis qu'une contrainte plus facile nous permettra de trouver une solution valide plus rapidement - ce qui à son tour nous permettra de couper des branches lors de l'exploration.

Pour évaluer l'impact de ce paramètre, nous avons étudié des variations du paramètre p sur l'ensemble des valeurs possibles, en comparant plus spécifiquement notre algorithme à l'algorithme SPC MDP sur le domaine du Business Jet (Algorithme 16 page 212).

Comme nous l'avons détaillé, le domaine du Business Jet décrit un avion de business ayant un plan de vol fixe. Cet avion est composé de n systèmes, pouvant chacun défaillir avec une probabilité

Taille	Fast-PCMDP (s)	SPC MDP (s)
3	0.02553085	0.0125
4	0.0402417	0.275
5	0.378344	2.815
6	1.2257	32.7899
7	4.88727	(dépassement de mémoire)

TABLE 5 – Comparaison des temps sur le domaine du Business Jet

donnée. Certains des aéroports du plan de vol ont l'équipement nécessaire pour réparer certaines de ces défaillances.

L'objectif pour l'avion est d'atteindre la fin du plan de vol avec un coût minimal de réparation, tout en assurant que la probabilité de certaines successions d'évènements catastrophiques ne dépasse pas un certain seuil : ceci représente la MEL (Minimum Equipment List), qui est un document légal restreignant les équipements défaillants avec lesquels l'avion est autorisé de décoller.

Le domaine augmente avec le nombre de systèmes ; cependant, il possède aussi de nombreuses contraintes PCTL qui nous permettent de couper de nombreuses branches lors de l'exploration.

Nous avons alors effectué une comparaison de Fast-PCMDP avec l'algorithme SPC MDP [SKTK14], dont les résultats sont visibles dans le tableau (Tableau 5 page 166). Cette comparaison est à l'avantage de Fast-PCMDP, puisque nous avons utilisé une relaxation des probabilités : dans SPC MDP, toutes les probabilités des contraintes sont fixées à 0 ou 1, mais les probabilités utilisées par notre algorithme vont de 0.1 à 0.9. Cependant, cette variation de probabilité donne une image globale du temps de calcul pour ce type de problèmes - en montrant en particulier que pour ce domaine en particulier les temps de calculs sont similaires sur l'ensemble des valeurs des probabilités des contraintes. Rappelons que l'algorithme SPC MDP est basé sur la programmation dynamique, mais n'utilise pas de fonctions heuristiques.

Les résultats particulièrement bons de notre algorithme peuvent être expliqués par le fait que ce domaine a des chemins clairement séparés, ainsi que des contraintes difficiles à remplir ; par conséquent, l'algorithme est capable de couper la plupart des chemins très tôt lors de l'exploration. Sur ce type de domaine, notre algorithme profite de sa capacité à sélectionner par priorité certains chemins par rapport à d'autres, au lieu de devoir attendre que tout l'espace d'états soit exploré pour calculer la validité de la politique comme dans SPC MDP.

VII.3.4 Conclusion et résumé des apports sur l'algorithme Fast-PCMDP

Dans les chapitres précédents, nous avons identifié plusieurs problèmes de décision dans le contexte des systèmes critiques pouvant être représentés sous la forme d'un modèle PCMDP. Plus précisément, nous avons établi que les modèles de type PCMDP semblaient être les seuls à apporter les garanties suffisantes en termes de respect des contraintes de sécurité dans le cas le plus général : bien que d'autres méthodes plus efficaces (en termes de temps de calcul) que PCMDP sont applicables dans la plupart des cas qui sont rencontrés dans un contexte industriel, il n'est pas possible dans le cas général de choisir des hypothèses qui nous permettraient de représenter le système dans un modèle plus simple que PCMDP. Ceci est en particulier le cas du modèle SPC MDP que nous avions développé dans une partie précédente.

Pour pallier ce problème, nous avons dans ce chapitre réalisé les apports suivants :

- Nous avons appliqué des concepts de recherche heuristique au problème PCMDP pour obtenir un **algorithme de résolution** efficace.
- Nous avons prouvé que cet algorithme était complet, c'est-à-dire qu'il permet d'obtenir la **politique déterministe valide et optimale** lorsqu'elle existe.
- Nous avons étudié l'influence de plusieurs paramètres de l'algorithme, en particulier en proposant une **méthode de sélection des plans partiels** permettant d'obtenir des solutions avec une forme intuitive.
- Nous avons enfin évalué les **performances de l'algorithme** sur plusieurs cas d'utilisation, et

montré qu'il permettait d'obtenir une solution plus rapidement que les algorithmes PCMDP-ILP et SPC-VI sur les cas de test.

Dans le cadre du modèle de Gestion des Modes Dégradés que nous avons développé dans le chapitre précédent, cet algorithme est particulièrement intéressant puisque les heuristiques peuvent être adaptées pour favoriser le choix de procédures lors de la recherche ; ceci implique que le choix d'un algorithme heuristique est doublement justifié, puisqu'il permet à la fois de restreindre le nombre d'états explorés ainsi que le nombre d'actions à évaluer.

Il est cependant important de noter que nous n'avons proposé ici qu'une version très simple de l'algorithme : en effet, de nombreux algorithmes de recherche heuristiques existent, qui utilisent des concepts permettant d'obtenir des résultats bien plus performants ou bien plus flexibles ; on pensera notamment aux algorithmes *anytime* qui permettent d'obtenir à tout instant une solution, tandis que l'algorithme continue à effectuer une recherche en tâche de fond pour améliorer la solution, ou encore à la parallélisation de la recherche. Comme nous pouvons le voir, l'algorithme Fast-PCMDP se prête bien à ce type d'adaptations, qui peuvent potentiellement améliorer sa performance de plusieurs ordres de grandeur.

Malgré tous ces avantages, il est important de souligner que l'algorithme Fast-PCMDP n'apporte pas les mêmes garanties que l'algorithme SPC-VI : Fast-PCMDP n'obtient pas une réelle optimalité, puisqu'il se limite aux politiques déterministes, et le choix de certaines améliorations dans les heuristiques peut potentiellement nous faire sélectionner une solution non-optimale. De plus, le temps de résolution de l'algorithme Fast-PCMDP est difficilement prévisible : il est tout à fait possible qu'un changement mineur dans le modèle mène à un temps de résolution complètement différent, puisque l'algorithme aura donné la priorité à une branche plutôt qu'à une autre.

Dans le chapitre suivant, nous étudierons dans quelle mesure l'algorithme Fast-PCMDP peut être utilisé dans un processus outillé réel ; pour cela, nous étudierons un cas d'application avionique, inspiré de la chaîne de données nécessaire à la capacité d'atterrissage à haute précision RNP-AR.

CHAPITRE VIII

ÉVALUATION DU PROCESSUS OUTILLÉ AU TRAVERS DE LA CONCEPTION D'UN MODULE D'ESTIMATION DE CAPACITÉ POUR LA FONCTION D'APPROCHE RNP-AR

Sommaire

Dans les chapitres précédents, nous avons défini un nouveau formalisme, le modèle de Gestion de Modes Dégradés, en nous basant sur le constat que la notion de qualité de service était centrale dans plusieurs problèmes de décision dans le milieu avionique. En raison du besoin de garanties fortes, imposé par la nature critique du système, nous avons suivi une réflexion similaire à celle que nous avions effectuée sur le problème du Business Jet, et nous avons établi que le modèle PCMDP était adapté pour la représentation de ce formalisme. Ceci nous a amené, dans le chapitre précédent, à envisager de nouvelles hypothèses permettant de résoudre les problèmes PCMDP : en nous limitant aux politiques déterministes, nous avons ainsi pu utiliser des techniques inspirées de la recherche heuristique pour proposer un algorithme de résolution pour PCMDP qui est bien plus efficace que les algorithmes de résolution existants.

Dans ce chapitre, nous avons donc un objectif double : d'une part, nous souhaitons évaluer dans quelle mesure cet algorithme se prête bien aux problèmes de taille industrielle, et d'autre part nous souhaitons établir dans quelle mesure le formalisme de Gestion de Modes Dégragés est effectivement un outil adapté pour étudier des problèmes de décision sur des systèmes critiques. Pour cela, il nous faut dans un premier temps construire le processus outillé global autour du modèle GMD et de l'algorithme de résolution, de façon similaire à ce que nous avions effectué pour le problème du Business Jet ; puis, nous pourrons alors mettre en œuvre dans un second temps ce processus sur un cas d'application avionique.

VIII.1 Contexte de la fonction RNP-AR

Puisque nous souhaitons assister les processus de conception des systèmes, nous rappelons tout d'abord les détails suivants sur le type de processus que nous traitons :

- Nous nous plaçons dans le cadre de la **modélisation amont** du système.
- L'objectif pour le concepteur d'un système est **d'analyser la propagation des défaillances** dans le système - ce que nous proposons de faire sous la forme d'une analyse de la propagation des défaillances de service.
- L'objectif pour le concepteur est donc principalement de réaliser une **analyse des dépendances** entre différentes composantes fonctionnelles du systèmes ; l'objectif n'est pas dans un premier temps d'évaluer une architecture précise - bien que nous préciserons par la suite que l'outil que nous proposons permet dans une certaine mesure d'assister à cette phase.
- Le concepteur du système peut disposer ou non d'une **connaissance préalable**, formelle ou informelle, sur le système ; nous ne présagerons rien sur les données disponibles en amont.
- Les modèles saisis peuvent présenter une notion de **hiérarchie**.
- Il est attendu, de la part du concepteur du système, que les étapes de modélisation et de validation amont puissent être effectuées **rapidement** - ce que nous proposons par exemple en proposant des méthodes de saisie graphique et textuelle adaptées.
- En partant des données saisies, il est attendu que plusieurs calculs puissent être effectués, tels que (mais non limités à) l'extraction de **graphes de dépendance**.

L'objectif, à ce stade de notre réflexion, est d'évaluer dans quelle mesure le formalisme GMD permet d'assister la conception de systèmes sûrs et optimaux. Une des manières de réaliser cette évaluation consiste à imiter le processus de conception que pourrait suivre un ingénieur, ce qui nous amènera alors à être confronté aux types de problèmes qu'il peut être amené à rencontrer. Nous avons choisi d'imiter le processus de conception de la fonction **RNP-AR (Required Navigation Performance - Authorization Required)** : cette fonction consiste à permettre à l'avion d'effectuer une approche à haute précision lors de l'atterrissage, en mettant en jeu un certain nombre de critères sur l'état du système, tels que la présence de chaînes de données dissimilaires ou encore sur la précision des données de position. Ce type d'approche est largement documenté dans la littérature, notamment concernant les critères de qualités de service nécessaires [KD94], concernant les contraintes réglementaires [FAA] ainsi que sur la valeur ajoutée de ce type d'approche [wik].

VIII.1.1 Justification du choix de la fonction RNP-AR

De façon pratique, le choix de cette fonction a été principalement basé sur le fait que les documents techniques et les documents de conception de cette fonction nous étaient accessibles au moment de cette étude. En effet, il semble évident que la conception d'une nouvelle fonction aurait nécessité plusieurs mois d'étude pour s'approprier le système, ainsi que la présence d'experts des différents domaines concernés, comme c'est les cas pour la conception d'un système réel ; ceci n'étant pas envisageable dans le temps imparti à l'étude, il a donc été nécessaire de choisir une fonction existante et bien documentée.

Le choix de nous baser sur un système existant a pour avantage principal de nous avoir permis d'évaluer rapidement les points forts et les points faibles du processus outillé : le processus de modélisation complet a consisté en 2 semaines de compréhension des documents existants et quelques jours de capture du modèle dans l'outil graphique que nous présenterons dans les sections suivantes. Ce choix a en revanche comme inconvénient principal de ne pas permettre de mettre en avant certains des aspects les plus novateurs de notre approche, tels que les reconfigurations de fonction ou la connexion de notre formalisme avec d'autres modèles de l'ingénierie système.

En particulier, le terme de **reconfiguration** dans le contexte avionique a un sens assez limité par rapport au type de reconfiguration que nous pouvons proposer : à partir de notre modèle, nous pouvons envisager des situations où le système choisit volontairement de demander un changement de modes de certaines ressources, par exemple pour compenser la défaillance d'une autre ressource ; dans les systèmes actuels, cette idée de compensation se retrouve, de façon presque exclusive, au travers de

la notion de reconfiguration pour les commandes de vol [HC13] [CAW88], où certains paramètres des lois de commandes sont modifiées pour prendre en compte des défaillances ou des dégâts de la surface de l'avion. Ce type de reconfiguration est donc le plus souvent initié lors de la disparition d'un signal : si une sonde ne fournit plus de données, le système cherche à remplacer la donnée manquante au travers d'une estimation obtenue à partir des autres sources disponibles. En termes d'architecture, cela peut donc être représenté sous la forme d'une chaîne où un composant donné permet de réaliser une **consolidation** : plusieurs sources différentes fournissent une même qualité de service, et ce composant sélectionne à tout instant la meilleure source, voire fournit une qualité de service consolidée en se basant sur plusieurs sources de moindre qualité de service. Nous voyons donc que ceci peut être effectué dans notre formalisme par le changement de mode d'un seul composant, tandis que notre formalisme permet de prendre en compte des reconfigurations coordonnées à l'échelle du système complet.

À notre connaissance, il n'existe aucun document décrivant des reconfigurations coordonnées entre plusieurs ressources d'un système ; ceci implique que, à nouveau à notre connaissance, l'optimisation des décisions de changement de mode dans un système est triviale pour tous les systèmes existants ; notons qu'en revanche, concevoir les formules réalisant la consolidation, c'est-à-dire les modes des ressources de consolidation, est loin d'être trivial. Par conséquent, puisque sur les systèmes existants l'approche manuelle est faisable et suffisante, nous nous attacherons à montrer :

- que notre processus outillé permet de modéliser les systèmes existants dans le formalisme GMD,
- que cette modélisation permet de concevoir les systèmes existants avec une qualité égale ou supérieure sur un certain nombre d'aspects (critères d'optimalité et respect des contraintes) vis-à-vis des méthodes manuelles,
- que la connexion avec des algorithmes tels que Fast-PCMDP est faisable, permettant d'affirmer que notre processus outillé sera applicable à la modélisation de systèmes présentant des reconfigurations coordonnées, lorsque l'industrie avionique souhaitera concevoir de tels systèmes.

Le choix du système RNP-AR a été réalisé en partie pour travailler autour de ces limitations, et nous permettre de tirer des conclusions intéressantes sur notre processus : comme nous le détaillerons dans une section suivante, ce système présente un problème intéressant de consolidation de données à la source, qui nous permet de montrer la faisabilité de notre approche, à défaut d'en montrer la plus-value sur les aspects de reconfiguration automatique.

Notons aussi que nous avons choisi de ne représenter dans ce rapport qu'un sous-ensemble de la fonction RNP-AR. Ceci s'explique en partie pour des raisons de confidentialité, mais aussi par le fait que ce sous-ensemble est suffisant pour illustrer les conclusions auxquelles nous sommes arrivées sur l'applicabilité de notre méthode.

VIII.1.2 Étude d'un extrait de l'architecture fonctionnelle de la capacité RNP-AR

Comme nous l'avons évoqué précédemment, RNP-AR signifie Required Navigation Performance - Authorization Required ; ce terme désigne donc un type de procédure d'approche, reposant sur des niveaux de précision requis en termes de performances à certains passages clés de l'approche. L'intérêt de ce type d'approche est double : (1) il permet à un avion d'atterrir dans des zones à terrain chaotique, par exemple des zones montagneuses, tout en s'assurant que le pilote a suffisamment de maîtrise sur l'avion pour ne pas entrer en collision avec le terrain, et (2) il permet de réduire l'encombrement des aéroports (et d'économiser du carburant), en proposant des trajectoires d'approches plus courtes.

Détails sur les contraintes propres à l'approche RNP-AR

Une trajectoire d'approche est qualifiée de RNP-AR lorsqu'elle a une des deux caractéristiques suivantes :

- La précision garantie sur les paramètres de navigation (RNP) est inférieure ou égale à 0.3 NM (Nautical Miles) lors de l'approche, ou inférieure à 1 NM en cas d'approche manquée.
- La trajectoire présente une courbure dans le plan de vol après le dernier point de passage avant l'atterrissage (Final Approach Fix - FAF).

Le second point est en particulier différent des approches classiques (ILS ou LPV), qui nécessitent que l'avion soit aligné bien avant la piste. En termes de réglementation, ceci impose des niveaux de

performance élevés sur l'intégrité (i.e. les données sont fiables) et sur la continuité (i.e. il n'y a pas de perte de données), ainsi que des certificats particuliers pour le constructeur de l'avion, le personnel de l'aéroport et le personnel à bord. Les probabilités et marges d'erreurs précises sont définies dans différents documents (AMC 20-26, AC 90-101) [EASa] ;

Pour garantir ces performances, le système repose sur un certain nombre de systèmes de monitoring embarqués, à la fois au sein de chaque équipement et à l'échelle du système complet. Ainsi, le système GPS dispose par exemple d'un système RAIM (Receiver Autonomous Integrity Monitoring), surveillant l'intégrité de la position GPS, par exemple en s'assurant que le système reçoit des informations de 5 satellites et est capable de détecter et exclure les satellites en faute ; on parle plus généralement d'OPMA (On Board Performance Monitoring and Alerting) pour désigner l'identification des capacités minimales dont doit disposer l'avion en termes de surveillance et d'alerte. Ceci implique donc qu'il est nécessaire de prouver que le système a une capacité de surveillance et d'alerte suffisante à chaque instant de l'approche.

De plus, les règlementations imposent des conditions sur les systèmes impliqués dans la capacité RNP-AR ; ceci comprend notamment :

- un système GNSS fonctionnel (capacité à gérer des données GPS),
- un système FMS (Flight-Management) avec une capacité BARO VNAV (fonction de pilotage automatique selon l'axe vertical reposant sur une altitude barométrique), ce qui implique de devoir disposer d'un système de gestion des données barométriques ADC (Air Data Computer),
- un pilote automatique fonctionnel,
- un système de navigation de secours (Inertial Reference System - IRS, Attitude and Heading Reference System - AHRS, Instrument Landing System - ILS) utilisé en cas de perte GPS et devant avoir le même niveau de performance de navigation,
- un système d'alarme pour informer le pilote en cas de déviation excessive ou de la perte de capacité.

À partir d'un certain niveau de précision (RNP < 0.3 NM ou approche interrompue < 1 NM), il est de surcroît demandé que le GNSS, le FMS et l'Air Data soient doubles - le système inertiel pouvant être unique.

Détails sur le sous-ensemble modélisé

Ces contraintes nous montrent que l'aspect critique du système RNP concerne la *chaîne de données*, dans la manière dont les données de position et d'attitude [1] sont acheminées depuis les capteurs extérieurs (sondes) jusqu'aux systèmes de guidage : l'ensemble des contraintes de redondance et de dissimilarité que nous avons évoquées ont été mises en place pour garantir que la position arrive au bout de la chaîne avec une certaine qualité.

Nous nous sommes donc limités dans notre évaluation aux systèmes composant cette chaîne de données. Précisément, le modèle est constitué de deux chaînes dissimilaires (Figure 31 page 174), correspondant à deux chaînes FMS (Flight Management System). Chacune des chaînes débute par une capture de l'information depuis les différentes sources de position (IRS 1/2/3 et GPS 1/2), puis l'information est divisée entre deux équipements dans chacune des chaînes : un équipement FMS effectue le calcul des données de guidage tandis qu'un second équipement "FMS monitoring" effectue le même calcul à des fins de surveillance. Ce calcul repose sur le passage par plusieurs fonctions, telles que le HPATH (chemin horizontal), le Flight Guidance (le calcul des paramètres de guidage), le Flight Director (la création de l'interface d'affichage du guidage à destination du pilote), avant enfin d'arriver aux PFD 1/2 (Primary Flight Display) qui correspondent aux écrans présents des deux côtés du cockpit.

En plusieurs points de ces deux équipements FMS, il existe des comparaisons avec l'équipement FMS de surveillance ; en revanche, le seul point de synchronisation entre les deux chaînes FMS globales est une synchronisation pour la mise à jour du plan de vol : lorsqu'un plan de vol est mis à jour sur l'une des chaînes, au travers du MCDU (Multifunction Control Display Unit), l'autre chaîne se synchronise en suivant une logique garantissant que le système ne se retrouve pas dans un état incohérent. Notons

1. direction de l'aéronef, dans le domaine aéronautique

aussi que les équipements FMS monitoring sont légèrement différents, puisqu'ils calculent des données différentes telles que l'enveloppe de vol, permettant une surveillance supplémentaire par rapport à la simple comparaison avec l'équipement principal.

Notons que, pour raison de confidentialité, nous ne pouvons reproduire ici ni le schéma complet du modèle représenté, ni les logiques de calcul et de vérification, ainsi que certains détails de l'interface graphique utilisée pour capturer le modèle. Nous ne présenterons donc que des vues haut niveau ainsi que des captures d'écran parcellaires (Figure 42 page 228). Néanmoins, les détails du modèle ne sont que peu pertinents pour l'évaluation globale, qui porte sur le processus de conception et non sur l'objet conçu au final.

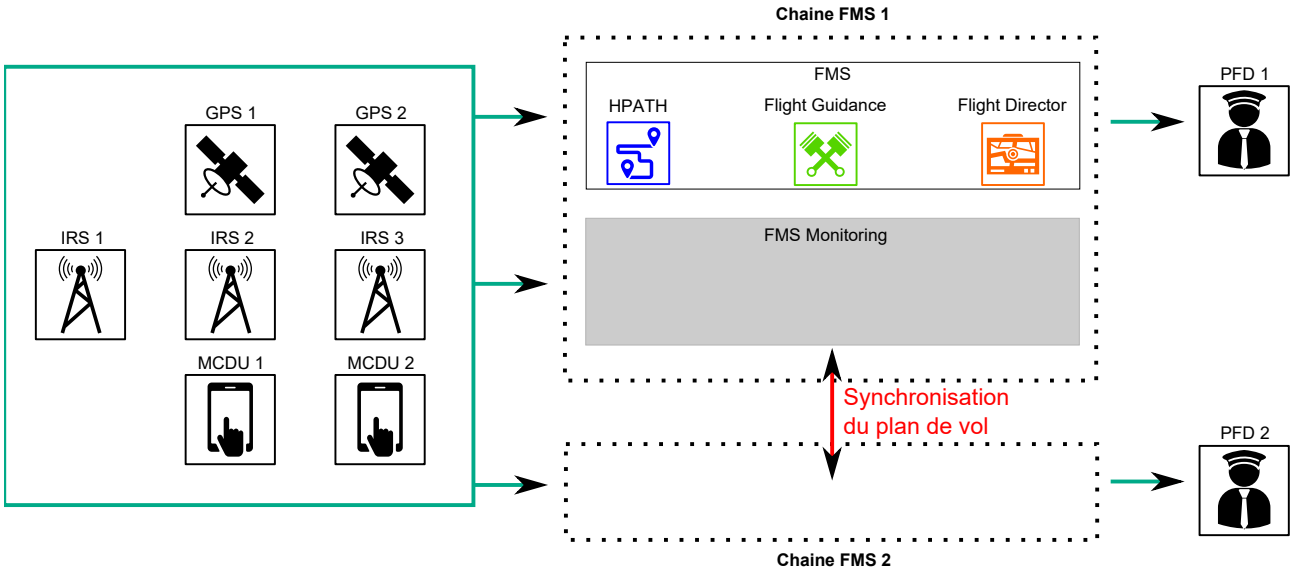


FIGURE 31 – Schéma des différents constituants de la chaîne de données de la fonction RNP-AR

VIII.2 Construction du processus outillé

À ce stade de notre réflexion, nous disposons d'un cas d'étude et d'un objectif général que le concepteur du système souhaite atteindre - celui d'analyser la propagation des défaillances dans le système en termes de qualités de service. Sur le problème particulier du RNP-AR, nous pouvons préciser ces objectifs :

- Le concepteur du système souhaite réaliser des **analyses de sécurité** sur le système.
- Il souhaite obtenir des **logiques de reconfiguration**, permettant au système de changer de comportement suite à des évènements extérieurs.
- Il souhaite obtenir une **Minimum Equipment List**, c'est-à-dire une liste minimale d'équipements devant être fonctionnels pour que l'avion soit autorisé à décoller, voire plus précisément soit autorisé à réaliser l'approche RNP-AR.
- Il souhaite pouvoir **connecter à d'autres outils existants** la représentation du système selon le formalisme GMD.

Ces différents objectifs permettent, dans le cadre de ce problème particulier, de construire un processus outillé tel que représenté sur le schéma (Figure 32 page 175) : en partant d'une saisie dans un démonstrateur, il est possible de réaliser un modèle GMD, sur lequel on peut réaliser successivement une traduction en langage Altarica, une vérification de contraintes de sécurité, une génération de conditions MEL puis une génération de tables de reconfiguration. Notons qu'il n'est pas nécessaire de réaliser ces opérations dans cet ordre - il est possible de générer les conditions MEL sans s'appuyer sur les résultats de la vérification de contraintes de sécurité - mais il est logique dans le cadre de la conception d'un système de réaliser ces étapes ainsi.

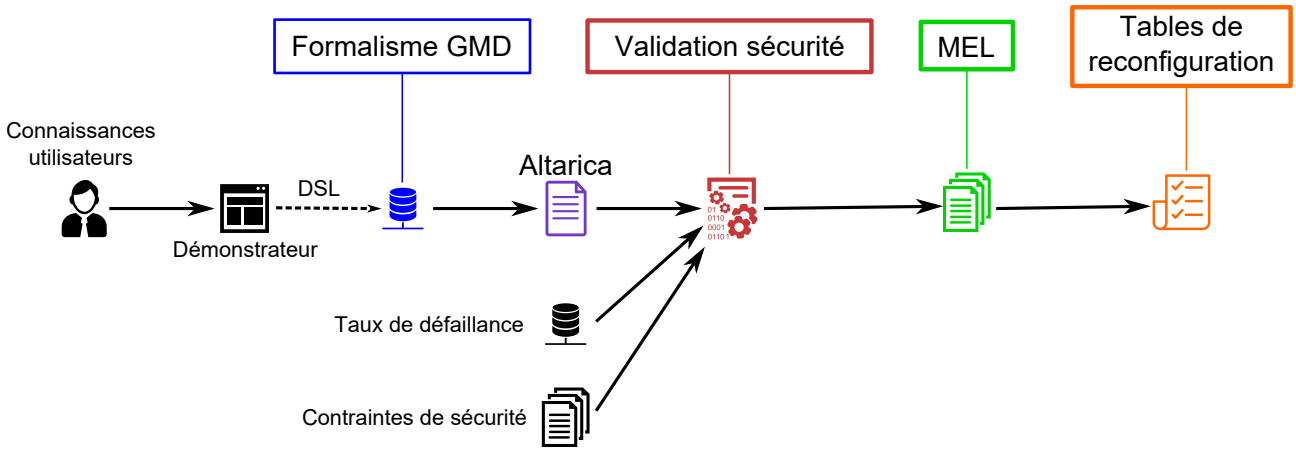


FIGURE 32 – Processus outillé construit sur le formalisme GMD.

Dans les paragraphes suivants, nous détaillerons alors chacune des étapes de ce processus, ainsi que leurs implémentations concrètes dans le contexte industriel, telles que nous les avons réalisées.

VIII.2.1 Implémentation d'un outil de capture portant le modèle de gestion de modes dégradés

Comme nous l'avons évoqué précédemment, nous souhaitons baser notre évaluation sur la capture d'un système avionique existant. De façon similaire à ce qui avait été réalisé pour le problème du business jet, ceci implique donc de construire un processus outillé permettant de capturer toutes les données nécessaires au problème, de les formaliser puis de les traduire dans le formalisme GMD. Bien que certaines données puissent être obtenues avec une capture automatique, par exemple une liste de ressources ou une liste de liens de communication au travers des tables décrivant les entrées et les sorties de chaque fonction, la plupart de ces informations sont beaucoup trop précises par rapport à la modélisation haut-niveau que nous envisageons : puisque nous cherchons à exprimer des implications logiques sur la capacité de l'avion à réaliser une approche RNP-AR, il n'est pas nécessaire de capturer ici l'ensemble des signaux qui circulent dans le système, ce qui augmenterait inutilement le nombre de

variables à traiter.

Nous souhaitons en premier lieu capturer les informations suivantes :

1. L'ensemble des ressources (fonctions) impliquées dans le périmètre (Figure 31 page 174).
2. L'ensemble des modes de fonctionnements ou de défaillances de ces ressources.
3. L'ensemble des services et des qualités de service impactant la capacité d'approche RNP-AR.
4. Les relations logiques liant ces qualités de service.

Notre moyen d'acquisition principal est donc la capture manuelle depuis la connaissance d'un utilisateur - c'est-à-dire dans notre cas la capture manuelle depuis des données non formatées telles que des schémas d'architecture ou des spécifications non formelles. Le processus outillé global doit donc prendre en compte cet état de fait et être construit pour faciliter la capture manuelle. Le retour d'expérience que nous pouvons proposer sur la construction de ce processus outillé est constitué des trois points suivants :

- La prise en compte de **principes d'ergonomie** améliore la qualité du modèle capturé.
- **L'automatisation de la représentation graphique** permet à la fois une plus grande maintenabilité du modèle et une capture globalement plus rapide.
- L'utilisation d'un **Domain Specific Language** multiplie les possibilités d'échanges avec les outils existants et les partenaires extérieurs tels que l'avionneur.

Principes d'ergonomie

Le domaine de **l'ergonomie des interfaces homme-machine** est largement documenté dans la littérature, à la fois en termes de ce qu'il faut faire pour augmenter l'efficacité de la communication entre l'homme et la machine [Shn92] et en termes de réalisation de cette augmentation [KP88]. En effet, de nombreuses méthodes existent pour évaluer l'ergonomie d'une interface [SB97], en particulier pour évaluer dans quelle mesure une interface peut être mise en œuvre avec facilité ou mise en œuvre par un nouvel utilisateur. Ces travaux mettent en valeur le fait qu'une interface efficace apporte des bénéfices autres que des bénéfices esthétiques : en soulignant au travers de l'interface comment les concepts doivent être manipulés, un outil bien conçu permet d'obtenir des résultats plus rapidement et avec une meilleure qualité. Dans notre cas, ceci se traduit de façon concrète par des modèles plus succins, présentant des dénominations claires et précises, ainsi que par une réduction des risques d'erreurs de capture.

Outre les principes d'ergonomie classiques, nous avons obtenu de très bons résultats en construisant une interface selon des règles de design connues du public, par exemple les règles du **material design** appliquées par l'entreprise Google [Goo] : en rapprochant l'interface de capture des interfaces connues des utilisateurs dans leur quotidien, nous avons pu nous rendre compte que la capture des modèles était effectuée avec une plus grande facilité, en particulier puisque l'interface pouvait être naviguée de façon intuitive (icônes et emplacements des panneaux de navigation connus) et puisque les concepts de superposition des informations amélioraient globalement la lisibilité du modèle (informations apparaissant dynamiquement au passage de la souris, mise en valeur de chaque étape du processus au travers de panneaux sur des niveaux différents,...).

Enfin, le second facteur indéniablement positif que nous avons pu expérimenter est l'utilisation du mouvement pour véhiculer des informations : plutôt que de représenter un lien entre deux ressources comme une flèche pour chaque service, comme il aurait été naturel de le faire dans des formalismes tels que UML, nous avons représenté ce lien au travers d'un trait sur lequel circule un service, c'est-à-dire sur lequel un point animé se déplace d'une ressource à l'autre. À nouveau, ceci participe à améliorer la lisibilité générale du modèle, et permet d'assister l'utilisateur dans sa représentation mentale au travers d'un visuel fort (Figure 33 page 177) : une ressource est un rectangle, un mode est une information entre crochets, un service est un rond en mouvement, et un changement de qualité de service est représenté par un changement de couleur du service.

Algorithme de disposition graphique automatique

En nous basant sur différentes lectures de la littérature abondante sur le Model-Based System Engineering [Est07], il nous est clairement apparu que la génération automatique de la représentation

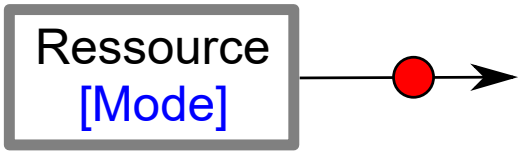


FIGURE 33 – Une représentation visuelle claire des concepts favorise la modélisation.

graphique d'un modèle était un gain mesurable pour la qualité du système produit. Ceci est appuyé par plusieurs arguments [Spe], en particulier sur le fait que plus un processus de modélisation est simple et rapide à mettre en œuvre, plus il a de chance d'être appliqué de façon correcte ; notamment, le fait de gérer les représentations graphiques et les différents diagrammes automatiquement à partir d'une représentation formelle unique représente un gain de temps en termes de création (il n'est pas nécessaire de s'intéresser à l'aspect esthétique), de mise à jour (il suffit de modifier la base de données centrale) et de validation (la vérification de la cohérence du modèle n'est réalisable que si le diagramme porte une certaine sémantique).

Bien qu'en marge des apports principaux de nos travaux, concernant les modèles de données et les algorithmes de résolution, ce type de réflexion est néanmoins essentiel lors de la construction d'un processus outillé complet, qui doit garantir à la fois la justesse des algorithmes et du formalisme utilisé, mais aussi la justesse des méthodes de capture des connaissances sous la forme d'un modèle. Nous nous sommes donc orientés vers les algorithmes permettant de représenter automatiquement des graphes (c'est-à-dire un ensemble de sommets reliés par des arêtes) de façon graphique, en nous intéressant à la fois aux outils open-source ou commerciaux disponibles [EGK+02], aux différentes classes d'algorithmes existants [BETT98] ainsi qu'aux critères permettant de définir ce qui est une esthétique favorisant la compréhension [PCJ96].

L'approche que nous avons choisie est du type **Force-directed graph drawing** [FR91] : l'idée est de s'inspirer de principes physiques pour fixer un certain nombre de contraintes entre les sommets d'un graphe, puis de mettre à jour le système jusqu'à stabilisation. Le modèle physique est que chaque sommet (pour nous chaque ressource) est une charge électrique positive, et chaque arête entre les sommets est un ressort. Les sommets se repoussent avec une force proportionnelle à l'inverse de la distance entre les deux sommets, ils ont donc naturellement tendance à s'éloigner les uns des autres ; cependant les sommets liées par des arêtes s'attirent avec une force proportionnelle au carré de la distance, donc ils auront tendance à ne pas s'éloigner trop les uns des autres. L'arrêt de l'algorithme est conditionné par une notion d'équilibre, soit par le fait que les positions des sommets ne changent pas d'une itération à l'autre soit par le fait qu'un critère d'énergie minimal est atteint.

Il a été montré au travers de plusieurs études que ce type de méthode a de nombreux avantages, tels que la mise en valeur des symétries du modèle, la simplicité de mise en œuvre sans connaissance particulière sur le graphe qui doit être dessiné, ainsi que les possibilités d'interactivité (modification en temps réel du graphe et réajustement des positions). En revanche, les deux inconvénients principaux sont la présence de minimums locaux de stabilité (le système se stabilise mais la représentation n'est pas optimale) ainsi que le temps de calcul qui est potentiellement long (la complexité est équivalente à $O(n^3)$) avant d'atteindre la stabilisation.

Dans la maquette que nous avons développée, nous avons réalisé un apport sur ces méthodes en intégrant une notion de hiérarchie : nous avons construit un algorithme, **Hierarchical Force-based Drawing**, qui réalise cette résolution avec des contraintes supplémentaires de proximité sur certains nœuds faisant partie d'un même sous-ensemble. Plus précisément, nous disposons d'un graphe constitué de nœuds (les ressources) nommés de façon hiérarchique, par exemple "$nom-du-grand-pere.nom-du-pere.nom$" ; ceci constitue donc un ou plusieurs arbres généalogiques entre les nœuds ; ces nœuds peuvent toujours être reliés par plusieurs liens, indépendamment de leurs positions dans la généalogie. Pour chaque nœud, on calcule une *taille de conteneur* à partir du nombre (récursif) de fils qu'il possède, ajouté d'une certaine marge fixe ; ceci correspond à une zone dans laquelle ses fils peuvent être disposés, avec le nœud père au centre de la zone. À chaque mise à jour, l'algorithme effectue donc récursivement une mise à jour des forces (d'attraction et de répulsion) et positions en partant des nœuds de plus haut niveau dans la hiérarchie (les plus grands ancêtres), et en imposant le fait qu'un nœud doit rester

dans les limites de la zone de conteneur de son père. De plus, lorsqu'un nœud père est déplacé, tous ses nœuds fils sont entraînés avec lui.

La représentation physique de ce phénomène pourrait correspondre à un ensemble de boîtes qui contiendraient d'autres boîtes et des billes aimantées roulant librement dans ces boites ; ainsi, au travers des contraintes de répulsion (aimants) et d'attraction (ressorts), les boîtes de plus haut niveau se retrouvent disposées dans l'espace en fonction des liens entre les billes qui sont contenues à l'intérieur de ces boîtes, tandis que les billes elles-mêmes sont réparties au sein de chaque boîte sur les bordures extérieures dans la direction générale des liens sortants.

Comme nous pouvons l'imaginer, ceci nécessite une notion de collision entre les conteneurs ; cette collision doit de plus être implémentée selon un cercle centré sur chaque nœud père : toute autre forme qu'un cercle favoriserait les blocages entre deux zones de collisions devant se rendre dans deux directions opposées. Notons que ceci ne présage en rien de la forme des zones de conteneur, qui peuvent par exemple être des rectangles inscrits dans le cercle de la zone de collision.

Cet algorithme a produit des résultats satisfaisants sur le plan esthétique, et présente dans l'ensemble les mêmes avantages et les mêmes inconvénients que les autres algorithmes de force-directed graph drawing. Deux inconvénients se trouvent néanmoins accentués : (1) avec l'ajout des nouvelles contraintes de collision et de zones, le système présente de nombreux minimums locaux, ce qui a nécessité la mise en place d'un mécanisme de relance aléatoire similaire à ce qui peut être fait pour des algorithmes de recuit simulé et (2) pour obtenir un résultat plaisant sur le plan esthétique, les zones de conteneur doivent être créées assez larges, ce qui implique que le dessin final occupera plus d'espace que le dessin obtenu avec une méthode classique.

Notre retour d'expérience sur cet algorithme est qu'il est particulièrement pertinent lorsque le système présente des relations hiérarchiques fortes, par exemple si des ressources appartenant à deux sous-systèmes ne communiquent jamais, ou communiquent nécessairement par l'intermédiaire de la ressource père. Pour ce type de système, le gain en termes de lisibilité et maintenabilité d'une vue hiérarchique est évident (comparé à une vue "à plat" comme sur la capture d'écran (Figure 42 page 228)), puisqu'il permet de concentrer la capture ou la modification du modèle sur un sous-ensemble du modèle complet.

Principes d'implémentation d'un Domain Specific Language

Enfin, bien que le modèle que nous avons étudié n'a pas nécessité de connexion avec des outils extérieurs, nous avons choisi de faire porter notre modèle par un **Domain Specific Language** (DSL), c'est-à-dire un langage dont les mots-clés et la syntaxe ont été choisis spécifiquement pour faciliter la compréhension par un utilisateur expert du domaine. Il existe plusieurs outils publics permettant de créer un tel langage, construits notamment à partir de UML [Sel07], d'un éditeur de langage informatique classique [Gro09] ou encore d'une grammaire formelle [Par07].

Nous ne détaillerons pas ici le langage que nous avons choisi, puisque sa syntaxe et sa sémantique ont peu d'intérêts dans le contexte de cette étude, mais il est néanmoins intéressant de noter que l'utilisation de DSL dans l'industrie est réalisable, à un coût d'investissement faible, et avec potentiellement des gains importants en termes d'aide à la capture de connaissances. Nous renvoyons le lecteur intéressé aux études que nous avons évoquées sur les principes et méthodes d'implémentation d'un DSL.

VIII.2.2 Évaluation de l'impact des défaillances au travers de la propagation des modifications de qualité de service

Dans les paragraphes précédents, nous avons détaillé les principes qui ont été utilisés pour la mise en place d'un processus outillé permettant de capturer des modèles selon le formalisme de Gestion des Modes dégradés. Ceci nous a permis - en appliquant des principes d'ergonomie et de Model-Based System Engineering, qui facilitent la capture depuis des connaissances d'experts - d'obtenir un modèle d'une chaîne de données associée à la fonction d'approche RNP-AR. Dans les paragraphes suivants, nous allons à présent étudier le type de conclusions qu'il est possible de tirer à partir de ce modèle.

Le premier résultat que nous pouvons obtenir est l'évaluation de la qualité de service à la sortie de la chaîne de données. En effet, dès lors que l'on connaît les dépendances logiques entre les qualités de

service en sortie de chaque composant, il suffit de propager la mise à jour de ces qualités de service dans le système pour obtenir la valeur exacte de la qualité de service en bout de chaîne.

Détails sur l'algorithme de propagation

Pour cette propagation, nous nous sommes basés sur la procédure de mise à jour que nous avions détaillée dans un chapitre précédent (Algorithme 11 page 132). Celle-ci consiste à appliquer successivement les règles de mise à jour sur les entrées et sorties, affirmant (1) qu'en cas d'inconsistance des entrées, toutes les qualités de service sortantes sont au minimum, (2) que si toutes les données d'entrée sont consistantes, alors les qualités de service sortantes sont fixées par le mode de la ressource et (3) que si plusieurs sources sont disponibles pour un même service, la qualité de ce service est automatiquement la plus élevé.

Il est cependant important de souligner que nous pouvons choisir deux modèles différents pour la capacité RNP-AR : un modèle où la règle (3) est effectivement appliquée, et où la sélection de la source qui a la qualité la plus élevée est toujours privilégiée, ou un modèle "en aveugle" où la sélection de la source doit être décidée par le système, en se basant potentiellement sur une information parcellaire. La seconde option peut sembler plus réaliste, puisqu'elle nécessite d'ajouter au modèle la transmission de messages de surveillance pour avertir chaque ressource des qualités de service reçues, cependant cette contrainte de réalisme est en vérité superflue dans notre cas puisque les logiques de sélection de sources ne sont pas basées sur la précision mais sur la comparaison à une référence : les qualités de service que nous propageons ne sont pas des qualités de service décrivant si la position est précise ou non, mais des qualités de service **d'intégrité**, c'est-à-dire décrivant si la position est fiable ou potentiellement erronée suite à une défaillance, ainsi que le niveau de **continuité**, c'est-à-dire le maintien du service indépendamment des événements pouvant se produire.

Prenons un cas pratique pour préciser ce point essentiel : considérons trois sources GPS, transmettant un signal de position à une ressource FMS fictive ; si on raisonne en termes de précision, et que l'on dispose d'un degré de précision sur chaque source, la ressource FMS peut réaliser une consolidation à la source et sélectionner en temps réel le GPS qui est le plus précis ; ceci n'est pas très réaliste : dans des cas réels, il n'est vraisemblablement pas envisageable de disposer réellement de l'information de précision sur chacune des sources - ou en tout cas, d'accorder suffisamment de fiabilité à l'information de précision pour baser la décision de la sélection de la source uniquement sur cette information.

À l'inverse, si on raisonne en termes d'intégrité, alors : si les trois sources fonctionnent de façon nominale, le FMS a une position en entrée nominale ; si une seule source fonctionne de façon erronée, le FMS choisit la valeur en fonction de la majorité et a donc toujours une position en entrée nominale ; si 2 sources fonctionnent de façon erronée, alors il est tout à fait possible que les deux sources erronées produisent la même valeur erronée, et donc le FMS a une position en entrée qui est elle aussi erronée. Nous voyons donc que contrairement au raisonnement en termes de précision, il ne s'agit pas d'une logique de consolidation à la source (un des GPS a toujours une valeur nominale, mais la valeur de la qualité de service entrante est considérée erronée), et que cette logique ne repose pas sur une connaissance supplémentaire sur la qualité de service (le FMS ne sait pas quel GPS fournit une qualité de service erronée).

Notons que, dans notre formalisme, il est nécessaire de créer une nouvelle ressource en entrée du FMS pour porter ces relations logiques plus complexes que de la consolidation en entrée ; cependant, nous avons choisi de faciliter la saisie de ce type de logiques dans l'outil de capture en permettant à l'utilisateur d'exprimer directement des logiques plus complexes sur les paramètres d'entrée, ce qui correspond d'une certaine manière à la création de "variables locales" dans les ressources.

En pratique, ceci a donc nécessité un pré-traitement (parsing d'une chaîne de caractère au travers d'un algorithme de parsing tel que l'algorithme du Shunting-Yard de Dijkstra [Dij]) ; notons aussi qu'au travers d'un parcours simple de graphe, il est possible de choisir un ordre de propagation des modifications dans le système qui minimise le nombre d'opérations effectuées lors de la mise à jour, c'est-à-dire dans notre cas de choisir un ordre de mise à jour qui va depuis le début de la chaîne (GPS, IRS) vers la fin de la chaîne (PFD).

Cette mise à jour est rapide, puisqu'elle prend moins de 5 milli-secondes pour chaque changement ; néanmoins, comme nous l'avons déjà évoqué précédemment, si nous permettons d'écrire des équations

logiques plus complexes entre des entrées et des sorties (ou entre des entrées et des variables intermédiaires) il est possible d'obtenir des situations où l'algorithme de mise à jour ne converge jamais, auquel cas il est nécessaire de remonter une erreur ou de mettre en place un compteur TimeOut pour arrêter la mise à jour. Ce risque (de non-convergence de la mise à jour) ne dépend que du type d'équation qui est permis, en particulier sur le fait que les équations conservent une notion de décroissance à chaque mise à jour.

Détails sur les qualités de service et les modes

Comme nous l'avons évoqué précédemment, nous nous intéressons principalement aux informations d'intégrité et de continuité dans les qualités de service ; l'intégrité est caractérisée par deux valeurs possibles pour la qualité de service : "Reliable" et "Erroneous". Puisque les modes de dysfonctionnement ne sont pas décrits dans les documents standards (les "modes de défaillances" qui sont présentés correspondent le plus souvent à des défaillances de service plutôt qu'à des défaillances de ressource), nous avons ajouté à la plupart des ressources un comportement par défaut entre deux états "Nominal" et "Failed".

Certains autres types de qualités de service ont été utilisés, par exemple pour les logiques de comparaison entre les différentes chaînes (messages "In-Fault" ou "No-Fault" envoyés au système) ou encore pour les logiques de mise à jour des plans de vol dans les FMS. Il est intéressant de noter que nous avons choisi des domaines de qualité de service à deux valeurs, mais que des domaines avec un plus grand niveau de détail sont possibles ; néanmoins, il n'existe pas actuellement d'information sur des domaines plus précis, puisque les documents et modèles disponibles dans l'industrie, par exemple dans le contexte de la sécurité, expriment les relations entre entrées et sorties sous la forme de logiques booléennes. Il est par ailleurs intéressant de noter que nous avons choisi des noms de valeurs plus détaillées que "vrai" et "faux" pour ces domaines, pour lever des ambiguïtés telles que lorsque l'utilisateur ne sait pas de façon intuitive si la valeur "vrai" pour un service de surveillance signifie que le système est dans un bon état ou si un message d'erreur est levé.

Notons enfin que, contrairement à certaines études de sécurité qui utilisent des domaines à trois valeurs "Ok/Erroneous/Ko", nous avons choisi de séparer ce type d'informations en deux domaines "Available/Not-available" et "Reliable/Erroneous" ; la raison de cette séparation est qu'il s'agit, à notre sens, de deux types de qualités de service différents : on décrit avec un domaine "Available/Not-available" le fait que le signal soit ou non reçu, tandis qu'on décrit avec un domaine "Reliable/Erroneous" le fait qu'il soit correct ou non. Ainsi, il est possible dans d'autres formalismes de combiner ces deux informations dans une variable à trois états - puisque lorsque le signal n'est pas reçu, alors il n'est pas important de savoir s'il est correct ou non - mais cette combinaison n'est plus possible dès lors qu'on souhaite associer une notion d'ordre dans les domaines : il n'est pas possible de dire que Erroneous est supérieur ou inférieur à l'état Ko dans le cas général. Ceci nous montre bien que la notion d'ordre dans les qualités de service constitue une spécificité de notre formalisme.

Exemple d'évaluation de la consolidation des données dans la double chaîne RNP-AR

Puisque nous avons choisi une représentation centrée sur l'intégrité et la continuité, nous pouvons obtenir des résultats qui concernent la propagation des défaillances d'intégrité et des défaillances de continuité. En particulier, nous pouvons dans une situation donnée évaluer si les données avioniques sont immédiatement disponibles et simuler des scénarios de défaillances pour étudier comment des pannes de ressource impactent la chaîne de données.

Nous avons illustré cette réflexion dans la figure (Figure 34 page 181) : il s'agit d'une capture d'écran d'une des fenêtres annexes de l'outil que nous avons développé, qui présente les relations de dépendance entre les différents services du système sous la forme d'une chaîne de dépendance : à droite de la chaîne se trouve un service particulier, puis la chaîne est construite en remontant (vers la gauche) de proche en proche les dépendances entre services entrants et services sortants, jusqu'à retrouver à gauche les modes des ressources, par exemple les modes de fonctionnement et de défaillances des GPS.

Cette représentation graphique est intéressante puisqu'elle montre de façon intuitive comment une défaillance en début de chaîne impacte un service en bout de chaîne : nous voyons sur cet exemple

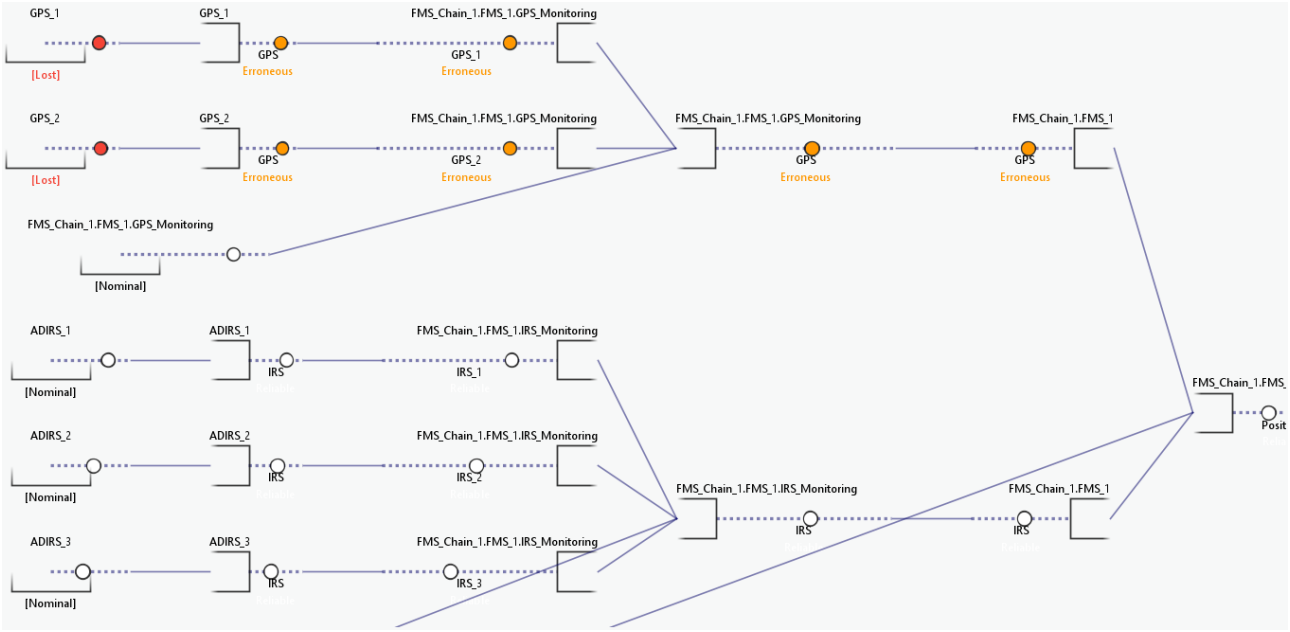


FIGURE 34 – Représentation graphique de la chaîne de dépendance et illustration de la propagation des défaillances de service.

qu'une panne de deux GPS provoque la propagation d'une valeur erronée jusqu'à un certain point du système, mais que le système a pu consolider le signal en un certain point (à partir des données IRS), ce qui fait que le service en bout de chaîne n'est pas impacté.

Comme nous pouvons le voir, nous avons donc obtenu, au travers d'une simulation de la propagation des défaillances de qualité de service, un outil permettant de déterminer dans quelle mesure notre système est résilient ou non aux défaillances. Nous verrons dans les paragraphes suivants que cette évaluation peut être obtenue de façon exhaustive et automatique en utilisant des outils de génération de coupes minimales.

VIII.2.3 Utilisation d'outils de génération de coupes minimales pour la validation de contraintes de sécurité

Dans les paragraphes précédents, nous avons construit un modèle de la chaîne de données utilisée par la fonction RNP-AR, centrée sur la propagation des défaillances de service en termes d'intégrité et de continuité. Nous avons montré qu'une simulation manuelle permettait d'obtenir des résultats intéressants, et surtout des résultats très visuels pour un utilisateur.

Nous pouvons aller plus loin en effectuant une exploration exhaustive du modèle : si nous simulions tous les cas de défaillance possibles, en produisant une table de toutes ces situations et des qualités de service de sortie, nous aurions alors en temps réel une estimation du risque de perte de service ; cette information peut être exploitée de deux manières : (1) lors de la conception du système, il est possible de construire plusieurs indices de mesure sur le système, qui indiquent si le système est plus ou moins résistant aux défaillances par exemple en termes de probabilités d'évènements redoutés, et (2) lors des opérations, il est possible de calculer en temps réel des indices de risque de pertes de service, permettant d'informer le pilote ou les systèmes qui peuvent anticiper les évènements en conséquence.

Cependant, la création d'une telle table n'est pas la méthode la plus efficace d'obtenir ce type de résultat. Nous pouvons ainsi nous baser sur les travaux réalisés dans le cadre de la sûreté de fonctionnement sur l'exploration des futurs possibles, en particulier dans le cadre de la recherche de **coupes minimales** : une *coupe* désigne "une combinaison d'évènements dont l'existence simultanée conduit à l'occurrence de l'évènement redouté"[Ade11] ; une coupe désigne donc un ensemble de défaillances de ressources tel que lorsque ces défaillances sont présentes simultanément une certaine condition est potentiellement remplie (par exemple ici la perte d'un service en sortie de chaîne) ; cette coupe est *minimale* si elle "ne contient aucune autre coupe", c'est à dire que lorsqu'on enlève une ou plusieurs

des défaillances de cette coupe, l'ensemble des défaillances restantes ne suffit pas à remplir la condition (le service de sortie n'est plus perdu).

Traduction du modèle dans le langage Altarica pour la connexion avec les outils Ocas et ARC

Il existe plusieurs outils de Model-Checking proposant de calculer automatiquement l'ensemble des coupes minimales à partir d'une représentation formelle du système, ce qui regroupe en particulier les solveurs basés sur Altarica (Cecilia Ocas, ARC [GV04],...) et NuSMV [CCG+02]. Afin de profiter des algorithmes efficaces présents dans de tels outils, nous avons donc réalisé une traduction de notre modèle formel dans le langage Altarica.

Cette traduction est réalisée en suivant les règles suivantes :

Règles de traduction du formalisme GMD en langage Altarica

1. Pour chaque domaine de qualité de service utilisé, on crée un domaine énuméré en Altarica ayant les mêmes valeurs.
2. Pour chaque ressource, on crée un nœud Altarica ayant le même nom.
3. Pour chaque service entrant et sortant, on crée une variable de flux du même nom (et du type correspondant in/out en Altarica Dataflow).
4. Pour chacun des services précédents, on fixe le domaine comme étant celui de la qualité de service, qui a été créé précédemment.
5. On crée une variable d'état "status" ayant pour domaine un ensemble énuméré constitué de tous les modes de cette ressource.
6. On crée les transitions appropriées entre les modes ; les noms des évènements sont les noms des modes préfixés par une chaîne de caractère connue.
7. On crée une variable de flux intermédiaire "consistance", à valeur booléenne vrai/faux.
8. On définit une assertion pour la variable consistance en fonction des critères d'entrée : $consistance = entree_1 \geq m^{in}(entree_1) \& \ldots \& entree_n \geq m^{in}(entree_n)$. Notons que l'opérateur $\geq$ peut être remplacé par une énumération de toutes les valeurs de qualité de service correctes.
9. Les assertions sont créées à partir d'une logique ternaire simple : si *consistance* est vrai, chaque sortie s vaut $m^{out}(s)$, dépendant du mode ; sinon chaque sortie vaut la valeur minimale de son domaine.
10. Enfin, un nœud principal est créé pour instancier les nœuds précédents et effectuer la liaison entre les services entrants et sortants.

Interprétation des coupes minimales pour la validation

À partir de la traduction du modèle dans le formalisme Altarica, et en nous appuyant sur des solveurs existants, nous pouvons alors réaliser une recherche des coupes minimales sur ce modèle, en prenant pour cible la perte de qualité de service en sortie de chaîne ; ces résultats peuvent être récupérés à nouveau, au moyen d'un parseur simple, pour être affichés de façon interactive dans l'outil. Pour une cible fixée, il est par exemple possible de calculer un majorant de la probabilité de défaillance d'une qualité de service en effectuant la somme des produits des probabilités des transitions présentes dans chaque coupe minimale. Cette méthode de calcul découle directement des méthodes classiques de sûreté de fonctionnement [Ade11], recommandées par les standards [ARP 4761], et suppose une indépendance des différentes transitions. Par conséquent, cette forme de validation des contraintes de

sécurité remplit un rôle similaire à ce qui peut être réalisé dans le cadre de la PSSA (Preliminary System Safety Assessment) [ARP 4761], mais ne dispense pas de réaliser d'autres analyses telles que l'analyse des causes communes.

Ce type de réflexions, classique dans les processus de validation de contraintes de sécurité, nous montre bien que le formalisme GMD permet de nombreuses possibilités en termes de validation amont (Early-Validation) : comme nous l'avions détaillé dans un chapitre précédent, l'objectif lors de la création du formalisme GMD était de permettre de représenter les capacités d'un système et d'évaluer l'impact de chaque action ou évènement extérieur sur ces capacités ; l'intuition était alors qu'avec cette représentation des capacités du système et de ses évolutions, il était possible de qualifier ce qui était une "bonne" ou une "mauvaise" décision, en fonction de contraintes de sécurité et des critères d'optimalité, à la fois en phase de conception du système et lors de son utilisation par le pilote ; nous pouvons voir ici que l'utilisation de techniques de Model-Checking nous permet effectivement de savoir en phase de conception si les choix d'architecture (fonctionnelle) sont "bons", en ce qu'ils respectent des contraintes de sécurité.

Nous pouvons illustrer ceci par un exemple académique : considérons une architecture fonctionnelle composée de deux ressources fournissant le même service, l'une des ressource étant la ressource principale et l'autre la ressource redondante (backup) ; on suppose que les deux ressources ont :

- une probabilité 10^{-X} par heure de vol (Flight Hour) de défaillir dans un mode où elles ne transmettent plus de signal, et
- une probabilité de 10^{-Y} de défaillir dans un mode où elles transmettent un signal erroné.

Alors, la probabilité de défaillance du service par heure de vol est de 10^{-2X} en continuité et de 10^{-Y} en intégrité : le service n'est plus émis si les deux ressources sont en panne ; le service émis est erroné si au moins une ressource est erronée, puisque le sélecteur ne pourra pas choisir quelle est la bonne source ; nous supposons ici que le sélecteur renvoie toujours un signal lorsque c'est possible. Notons qu'en toute rigueur, il est nécessaire d'ajouter la probabilité de défaillance 10^{-Z} du sélecteur de source, ce qui donne la formule $10^{-2X} + 10^{-Z}$ pour la continuité (évidente en se basant sur les coupes minimales), et la formule $10^{-Y} + 2 * 10^{-Z-Y}$ pour l'intégrité, puisque si le sélecteur n'est plus fiable, il est possible qu'il sélectionne la mauvaise source si une des sources n'est plus fiable ; ceci peut néanmoins être simplifié en 10^{-Y}, si on ne considère que les ordres de grandeur.

Si on représente une architecture à base de trois ressources, comme représentée dans la figure (Figure 35 page 184), alors il est nécessaire que les trois ressources soient en panne pour obtenir une perte de service en continuité (le service n'est plus rendu) ; ceci nous donne la formule suivante : $10^{-3X} + 10^{-Z}$. En revanche, en termes d'intégrité (le service rendu n'est plus fiable), il est suffisant que :

- au moins deux de ces ressources soient défaillantes pour obtenir une perte d'intégrité (le sélecteur ne sait plus quelle source choisir),
- ou qu'une ressource soit défaillante et qu'une des ressources restantes soit en panne,
- ou encore qu'une ressource soit défaillante et que le sélecteur soit défaillant.

Ceci nous donne la formule suivante : $3 * 10^{-2Y} + 6 * 10^{-Y-X} + 3 * 10^{-Z-Y}$. Notons que puisque nous ne faisons aucune hypothèse sur X ou Y, nous ne pouvons pas supprimer les termes les plus faibles, comme il est d'usage dans des cas réels.

Ici, le choix de l'architecture appropriée entre ces deux propositions peut donc être effectué en comparant les probabilités de perte de qualité de service en termes d'intégrité et de continuité - bien que la même démarche puisse être effectuée avec des types de qualité de service différents ; cette comparaison n'est en revanche qu'un des critères de décision, puisqu'il est par exemple possible de comparer les architectures en termes de coût : ajouter une nouvelle ressource redondante multiplie le coût financier, et agir sur les probabilités X et Y impacte aussi le coût puisque des équipements plus fiables sont d'autant plus cher à concevoir et produire.

Bien que réalisable, le calcul devient fastidieux dès lors qu'on envisage des cas plus complexes, par exemple des cas impliquant des ressources avec des taux de défaillance différents, ou des combinaisons de sélecteurs ou de comparateurs sur plusieurs étages. Ceci devient considérablement moins fastidieux lorsque le calcul est réalisé dans un outil tel que celui que nous avons mis en place : considérons par exemple une étude cherchant à savoir s'il est possible d'obtenir des résultats plus intéressants en

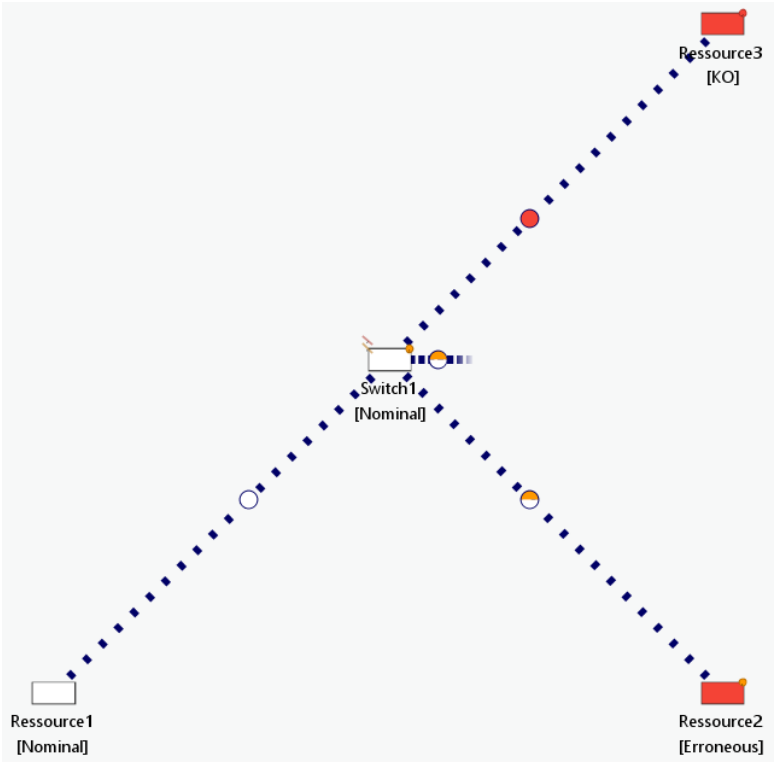


FIGURE 35 – Exemple académique d'une architecture à redondance double : si une ressource transmet un signal erroné et qu'une autre ressource ne transmet plus de signal, le sélecteur ne peut plus choisir et renvoi donc un signal potentiellement erroné.

augmentant le nombre de ressources. Ceci est en particulier avantageux si on considère des ressources ayant une plus faible fiabilité : on pourrait par exemple envisager de profiter du prix particulièrement bas des puces électroniques pour téléphones mobiles, qui n'ont évidemment pas les même garanties que les calculateurs conçus pour le milieu avionique, mais qui pourraient potentiellement compenser ce défaut au travers du nombre.

Réaliser un prototype de ce type d'architecture est facile au travers d'un outil de modélisation formel : un tel outil peut tout d'abord assister un ingénieur dans l'évaluation rapide du respect des contraintes ; mais il peut aussi directement générer un certain nombre de combinaisons possibles (variations sur le nombre de systèmes, sur le nombre de couches de sélecteurs, sur les probabilités) au travers du DSL, en couplant l'exploration des architectures possibles à un algorithme d'optimisation, par exemple du type Algorithme Génétique.

Ce type de méthodes, que nous n'avons que partiellement exploré, donne lieu à des architectures telles que celles représentées dans la figure (Figure 36 page 185) : si le taux de défaillance des ressources est $X = 1$, ce qui veut dire d'une certaine façon qu'elles doivent être certifiées pour au moins 10 heures de vol successifs, et que le taux de défaillance pour les sélecteurs est $Y = 9$, ce qui correspond à la probabilité la plus faible vérifiée actuellement lors des certifications (Extremely Improbable), alors des combinaisons de 9 systèmes en 3 étages donnent des probabilités de défaillance en continuité de $2 * 10^{-9}$ et des probabilités de défaillance en intégrité de $3.5 * 10^{-3}$; en revanche, si on conçoit un sélecteur pouvant évaluer 9 sources différentes, la probabilité de défaillance en continuité est de $2 * 10^{-9}$ et celle d'intégrité de $7.5 * 10^{-3}$.

Nous voyons donc qu'il y a un bénéfice non négligeable à sélectionner l'option étagée ; en particulier, ceci montre que pour l'intégrité il est profitable de minimiser le nombre de services impliqués dans chaque comparaison, bien que d'autres hypothèses que celles que nous avons prises sur les règles de propagation des défaillances d'intégrité peuvent donner des résultats différents. Par ailleurs, ces deux exemples montrent qu'une architecture jouant sur le nombre plutôt que la qualité est parfaitement viable, sous condition qu'il soit possible de garantir l'intégrité des données émises avec une probabilité suffisante, et potentiellement avantageuse en termes de coût de fabrication.

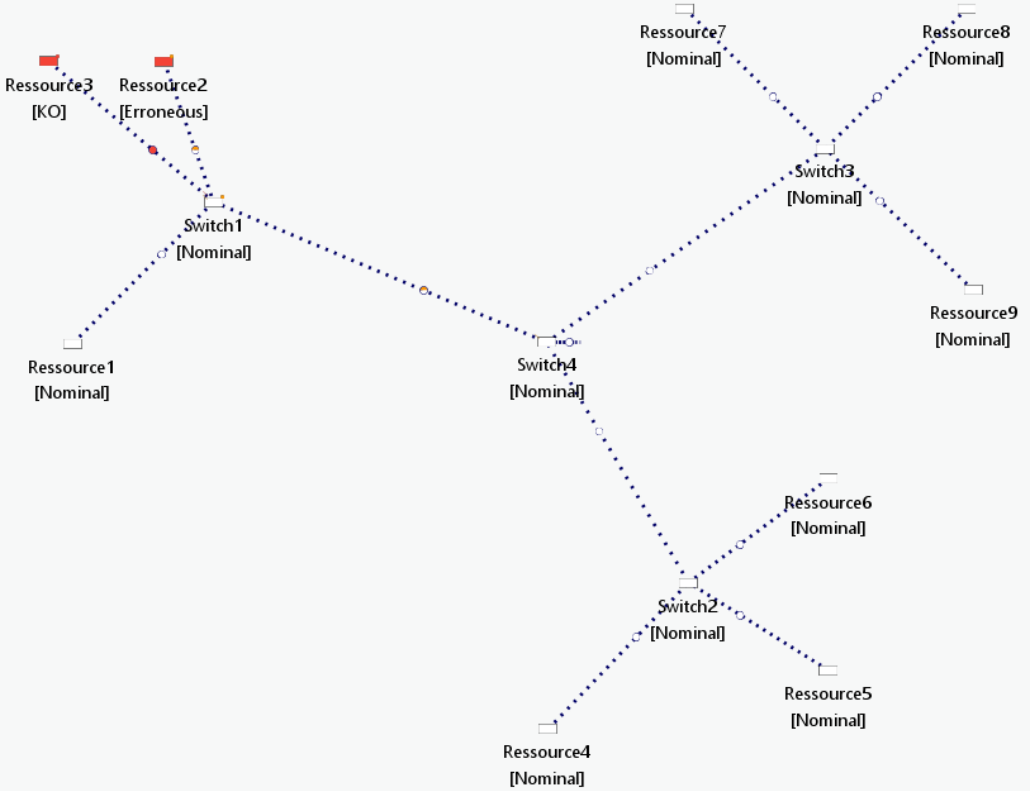


FIGURE 36 – Exemple académique d'une architecture à redondances étagées : avec l'hypothèse que le service est erroné lorsqu'une majorité de services entrants est erronée, cette architecture est plus avantageuse qu'une architecture à plat.

Il aurait évidemment été possible de réaliser ce type d'études dans d'autres formalismes ou d'autres outils que celui que nous avons utilisé ; cependant, cet exemple montre les deux points suivants :

- Les techniques de validation formelle, intégrées dans les outils de conception, apportent une plus-value réelle pour la prise de décision lors de la conception d'un système.
- Le modèle GMD est un formalisme capable d'effectuer la liaison avec des outils de validation formelle, en particulier lorsque l'on souhaite étudier plusieurs types de propagations de qualité de service dans un système.

VIII.2.4 Génération automatique de conditions MEL

Dans les paragraphes précédents, nous avons montré qu'il était possible d'assister les décisions prises lors de la conception au travers de l'utilisation de techniques de génération de modèle et de validation amont, en se basant sur le modèle GMD. Il est possible d'exploiter ces techniques pour obtenir des résultats utilisables aussi lors de l'utilisation du système par le pilote, par exemple en lui permettant d'évaluer l'état des capacités de l'avion au moment du décollage.

Plus précisément, nous pouvons générer une Minimum Equipment List, correspondant à la liste des équipements pouvant être défaillants au décollage sans que l'avion ne soit mis en danger. Pour générer cette liste, il suffit d'explorer de façon exhaustive toutes les configurations possibles, c'est-à-dire toutes les combinaisons de modes des ressources qui peuvent exister, et évaluer dans chacune de ces configurations quel est le risque de perte de capacités critiques : dans chacune de ces configurations, nous pouvons réaliser une opération similaire à la recherche de coupes minimales que nous avons décrite précédemment puis calculer la probabilité de perte des qualités de service critiques ; si ces probabilités de perte sont inférieures à un certain seuil, alors la configuration concernée est ajoutée à la MEL et l'avion sera autorisé à décoller dans cette configuration.

Il est possible néanmoins d'optimiser cette procédure en construisant un algorithme approprié, plutôt qu'en utilisant un algorithme de recherche de coupes minimales à chaque étape. Ceci implique en particulier (1) d'explorer les configurations possibles depuis l'état initial, au travers d'un parcours

en profondeur, (2) jusqu'à arriver dans une configuration où une des qualités de service critique est défaillante, puis (3) de propager la probabilité de configuration en configuration, jusqu'à (4) obtenir l'ensemble des configurations où la probabilité d'atteindre une situation redoutée est inférieure au seuil fixé. Cette résolution est donc en tout point similaire à celle qui peut être réalisée avec des algorithmes tels que [TKIS11].

Une attention particulière doit être portée aux transitions contrôlables : une première option (Figure 37 page 186) est de choisir une approche pessimiste, consistant à dire que le décollage n'est pas autorisé dès lors qu'une séquence problématique existe, qu'elle soit composée indifféremment de transitions contrôlables ou exogènes ; ceci revient à fixer une probabilité de défaillance pour les transitions contrôlables à 1, c'est-à-dire à considérer que l'opérateur prendra toujours la mauvaise décision. Dans ce cas, la MEL est plus restrictive que ce qu'elle pourrait être en réalité : il est possible qu'il existe des actions que l'opérateur puisse prendre qui permettraient à l'avion de décoller, mais nous ne prenons pas en compte cette possibilité.

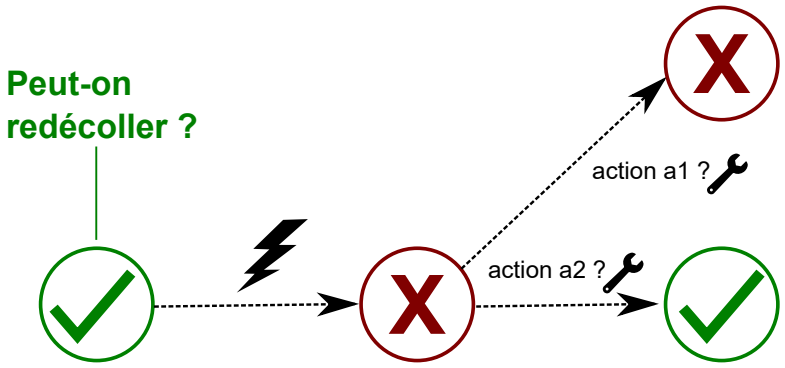


FIGURE 37 – Option 1 : avec une approche pessimiste, l'avion ne sera pas autorisé à décoller, parce qu'il existe une défaillance critique et que l'on suppose que la mauvaise action (a_1) sera prise.

La seconde option (Figure 38 page 187) est de ne considérer que les actions de reconfiguration immédiates : dans chaque configuration, nous considérons toutes les combinaisons d'actions possibles ; pour chaque combinaison d'action, on construit la configuration obtenue après avoir effectué cette combinaison ; on évalue ensuite la probabilité de défaillance de la qualité de service en fixant la probabilité des transitions contrôlables à 1 ; enfin, on sélectionne l'action de reconfiguration qui mène à la plus faible probabilité. Ceci est une approche du type "GO-IF" : on autorise le décollage sous condition qu'une certaine reconfiguration soit immédiatement effectuée, mais on garde une vision pessimiste de ce qui peut se passer une fois que l'avion a décollé. Notons cependant que dans les MEL actuelles, les conditions "GO-IF" contiennent aussi un certain nombre de tests à effectuer pour s'assurer que certains équipements critiques sont dans la bonne configuration, par exemple vérifier qu'une valve est bien fermée ou que la chaîne redondante fonctionne bien.

Enfin, la troisième option (Figure 39 page 187) est de choisir une approche optimiste, ou une approche automatisée : à chaque configuration explorée, on sélectionne la meilleure action, selon un algorithme similaire à l'algorithme Value-Iteration pour des processus décisionnels markoviens. Ce que nous cherchons à minimiser ici est la probabilité d'atteindre un état où une des qualités de service est défaillante. Cette vision est optimiste, puisqu'elle suppose que l'opérateur ou le système choisira toujours l'action de reconfiguration adaptée ; elle n'en est cependant pas moins réaliste : au travers d'une telle étude, il est possible d'informer l'opérateur de maintenance ou le pilote du meilleur choix possible, ainsi que des perturbations pouvant survenir et des scénarios de contingence qu'il faudra appliquer face à ces perturbations ; muni de ces informations, l'opérateur peut dans tous les cas choisir la prudence et décider d'effectuer des réparations supplémentaires.

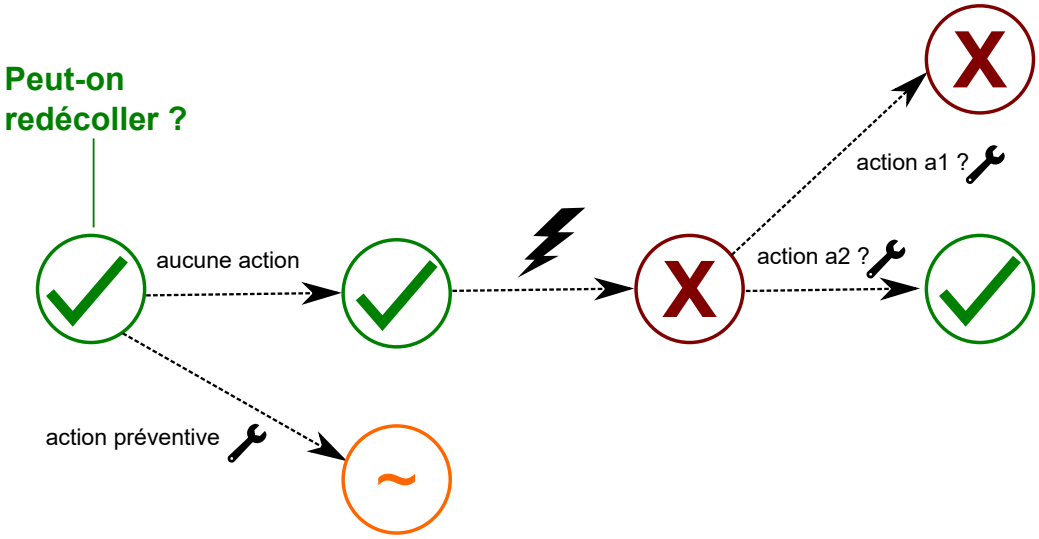


FIGURE 38 – Option 2 : avec une approche GO-IF, l'avion sera autorisé à décoller sous condition d'effectuer une action préliminaire, qui dégrade potentiellement ses qualités de service : sinon il existe une défaillance critique et on suppose toujours que la mauvaise action (a_1) sera prise.

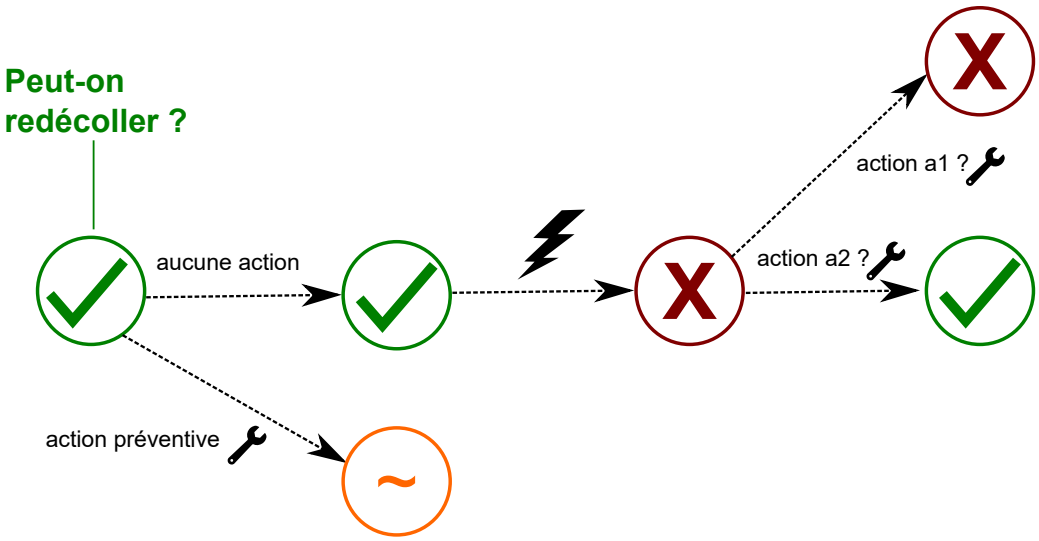


FIGURE 39 – Option 3 : avec une approche optimiste, l'avion sera autorisé à décoller sans conditions : on suppose que la meilleure action possible (a_2) sera toujours prise.

VIII.2.5 Génération de tables de reconfiguration au travers d'une traduction PCMDP

Enfin, il est possible d'évaluer l'impact de chaque décision de reconfiguration sur les capacités du système, c'est-à-dire d'informer un opérateur des modifications immédiates causées par un changement de mode d'une ou de plusieurs ressources, ainsi que de la probabilité de défaillance de certaines qualités de service suite à ce changement.

Pour une évaluation immédiate, le pilote (ou plus généralement l'opérateur) peut souhaiter interroger le système sur les conséquences de certaines actions, comme par exemple dans notre cas un changement de source de données ; il est alors facile de calculer une table associant à toutes les configurations possibles la valeur de certaines qualités de service identifiées, par exemple la précision de sortie, et de transmettre cette information au pilote sous une forme visuelle claire. Pour une évaluation à plus long terme, le pilote peut souhaiter savoir dans quelle mesure une décision qu'il prend immédiatement peut mener à une situation problématique dans le futur, problématique selon des contraintes de sécurité ou problématique en ce qu'une ou plusieurs défaillances futures peuvent amener le système à avoir des qualités de service bien plus faibles que ce qu'il aurait eu avec une autre solution . L'objectif est donc,

d'une certaine manière, de pouvoir proposer trois options au pilote :

- Les actions **interdites** : il s'agit d'actions dégradant la situation de l'avion ou le mettant en danger pour la suite de la mission, en amenant l'avion dans un état où il ne respecte pas les contraintes de sécurité.
- Les actions **possibles** : il s'agit simplement des actions qui ne sont pas interdites, c'est-à-dire qu'elles améliorent globalement l'état du système tout en ne le mettant pas dans une situation dangereuse.
- Les actions **optimales** : il s'agit des meilleures actions possibles, qui permettent au système d'avoir les meilleures qualités de service possibles à la fois à l'instant suivant la reconfiguration et lors des évolutions futures possibles. Ces actions participent donc à la fois à la performance de l'avion et à sa résilience face à de nouvelles défaillances.

Pour obtenir ces trois types d'actions, nous pouvons nous appuyer sur le formalisme PCMDP ainsi que sur l'algorithme de résolution que nous avons défini dans le chapitre précédent : en effectuant une résolution d'un problème PCMDP sur le modèle, nous pouvons obtenir pour chaque configuration possible la meilleure action ainsi que l'ensemble des actions valides ; en mettant cette information sous la forme d'une table, il est donc seulement nécessaire d'effectuer cette résolution au moment de la conception, puis d'embarquer à bord de l'avion une interface qui interrogera cette table à la demande du pilote.

Notons cependant que, lors de nos expérimentations, nous n'avons réalisé cette connexion entre le modèle GMD et l'algorithme de résolution PCMDP que sur des modèles académiques ; la raison de cette limitation, comme nous l'avons évoquée précédemment, est qu'il existe peu de modèles avionique prenant en compte des concepts tels que les changements de modes de fonctionnement coordonnés ainsi que leurs impacts sur les qualités de service - et aucun de ces modèles ne nous était facilement accessibles pour la réalisation de cette étude. En effet, sur des modèles tels que la chaîne de données que nous avons étudiée dans les paragraphes précédents, la seule problématique complexe est celle du choix d'une architecture favorisant la consolidation des données, tandis que la stratégie de choix des actions de reconfiguration en elle-même est triviale : chaque ressource choisit la source avec la plus grande qualité de service à tout instant.

Il est toutefois pertinent de noter que cette stratégie n'est pas triviale à appliquer - pour preuve plusieurs cas d'accidents ont été provoqués par un mauvais changement de source de la part du pilote, ou encore par l'extinction du mauvais système (côté gauche à la place du côté droit ou inversement), le plus souvent parce que le pilote se trouve dans une situation de stress.

Dans les paragraphes suivants, nous donnons quelques détails sur la manière dont la connexion entre le modèle GMD et l'algorithme de résolution Fast-PCMDP a pu être réalisée.

Adaptations de PCMDP aux besoins du modèle GMD

Lors de la connexion avec l'algorithme PCMDP, il est nécessaire de réaliser des ajustements : nous avons déjà évoqué précédemment le fait qu'il était possible de privilégier certaines successions de transitions contrôlables lors de l'exploration ; ceci peut-être fait naturellement au moment du choix d'une action à ajouter à la politique partielle, en ajoutant plusieurs transitions contrôlables successivement au lieu d'une seule ; nous pouvons implémenter ce mécanisme en explorant jusqu'à N transitions en avant, puis en commençant la génération des politiques partielles suivantes par l'ensemble des transitions menant au meilleur état immédiat. Nous pouvons par ailleurs éliminer immédiatement les combinaisons de transitions contrôlables qui n'ont aucun effet sur les qualités de service.

Nous pouvons cependant ajuster le critère de priorité entre les politiques pour favoriser - toutes les autres choses étant égales - les politiques partielles qui ont le moins de transitions contrôlables ; ceci permet non seulement d'éliminer les combinaisons contenant des transitions superflues, mais aussi de favoriser les combinaisons qui ne rendent pas plus complexes les reconfigurations suivantes : une suite d'actions A B est à préférer à une suite d'actions C D E si le résultat immédiat est le même (la qualité de service défaillante est récupérée) ; et une suite d'action A B est à préférer à une suite d'actions C D si lorsqu'une défaillance de la chaîne redondante survient la première séquence d'actions nous amène dans une situation où il est plus facile (en nombre d'actions) de reconfigurer le système.

Le second ajustement que nous devons faire concerne le choix d'un paramètre γ de dévaluation :

le choix d'une action suite à une défaillance est basé sur la valeur espérée, qui est la somme d'une récompense immédiate et d'une somme pondérée et dévaluée des valeurs espérées dans les états suivants, soit $R(s,a) + \gamma \sum_{s'} T(s,a,s')V(s')$. Ceci signifie qu'une action est choisie si elle remet immédiatement le système en état, et dans une moindre mesure si elle permettra de remettre le système en état dans le futur. Puisque les défaillances sont des évènements rares, inférieurs de plusieurs ordres de grandeur par rapport à la probabilité qu'il n'y ait pas de défaillance, nous souhaiterions que le poids des défaillances futures ait une importance minime dans la décision, par exemple qu'elles ne servent qu'à distinguer deux solutions identiques ; or, puisque nous avons construit la dynamique du modèle PCMDP sur un processus de saut, les probabilités des transitions qui sont utilisées dans le calcul de la valeur sont des ratios de taux de défaillance, c'est-à-dire potentiellement proches de 1.

Ceci est en partie compensé par le fait que la récompense immédiate contient une multiplication par le temps moyen passé dans cette configuration, mais un autre mécanisme de compensation possible est de fixer une valeur γ très basse : en fixant γ inférieure à la plus petite valeur de récompense non nulle, de plusieurs ordres de grandeur, nous permettons à l'algorithme de toujours sélectionner en priorité l'action remettant le système dans le meilleur état immédiat, puis en cas d'égalité entre deux actions de considérer les états futurs.

Notons par ailleurs que si plusieurs défaillances futures sont possibles, alors le plus souvent seules celles qui sont les plus probables en ordre de grandeur valent la peine d'être considérées : si une défaillance d'une ressource A a une probabilité d'occurrence de 10^{-3} par heure de vol et que toutes les autres ressources ont une probabilité d'occurrence de 10^{-6} par heure de vol, alors la valeur de la configuration peut se limiter à considérer la récompense immédiate ainsi que la valeur dévaluée venant de la défaillance de la ressource A. Il est donc possible d'optimiser l'algorithme pour prendre en compte ce fait, par exemple en ne calculant pas l'heuristique de valeur pour les évènements les plus rares.

Notons aussi qu'il est possible de rencontrer des problèmes de calcul flottant : lorsqu'on additionne des valeurs avec des ordres de grandeur très différents, certaines méthodes de calcul réalisent des approximations, par exemple en ne prenant pas en compte les dernières décimales ; pour remédier à cela, il est possible de choisir une comparaison des actions qui limite le nombre d'additions, par exemple en comparant dans un premier temps les actions possibles en fonction de la récompense immédiate, puis en cas d'égalité en comparant la partie de la valeur amenée par l'évènement le plus probable, et ainsi de suite jusqu'à pouvoir départager deux actions ; un tel opérateur peut aisément remplacer l'opérateur *max* utilisé traditionnellement dans les étapes de Value Iteration. Ce type de considération a déjà été évoqué précédemment, et nous pouvons noter à nouveau ici que des modèles de préférence plus complexes [JM+09] nous semblent parfaitement adaptables au modèle GMD et à l'algorithme de résolution Fast-PCMDP.

Implémentation des heuristiques à partir de la forme particulière du problème

Dans le chapitre précédent, nous avions détaillé une méthode générale pour obtenir des heuristiques sur des problèmes exprimés en formalisme STRIP ; bien que cette méthode soit toujours applicable dans notre cas, elle n'est cependant pas la meilleure : en concevant des heuristiques plus précises, qui se basent sur la forme particulière du modèle GMD, il est possible d'obtenir une performance accrue lors de la résolution.

L'algorithme de résolution repose sur deux heuristiques : une heuristique pour la valeur ainsi qu'une heuristique portant sur les probabilités d'arriver dans un état redouté ; pour la première heuristique, il semble évident qu'il est intéressant de **se limiter à la récompense immédiate**, c'est-à-dire limiter la valeur $R(s,a) + \gamma \sum_{s'} T(s,a,s')V(s')$ à la première composante R ; l'heuristique obtenue pour un état non étendu est alors le maximum sur les actions possibles de cette valeur $R(s,a)$.

Ceci ne donne pas en revanche une heuristique admissible dans le cas général, puisque la valeur de l'heuristique est inférieure à la valeur réelle ; pour pallier ce problème, nous pouvons simplement choisir comme heuristique $R(s,a) + 1$, lorsque γ a été fixée de façon à ce que le terme suivant soit inférieur ou égal à 1, qui est bien supérieur pour tout état à la valeur réelle. L'implication principale de cette majoration est que cela favorisera un peu plus l'exploration que si nous n'avions pas ajouté le terme +1, par exemple dans le cas rare où il aurait le choix entre une action l'amenant dans un état exploré et une action l'amenant dans un état non exploré mais ayant la même récompense immédiate.

La seconde heuristique, portant sur les probabilités de respect des contraintes de sécurité, peut de façon intuitive être obtenue à partir d'une **recherche des coupes minimales à l'état initial** : en effectuant un calcul des coupes minimales dans l'état initial, on obtient une estimation des chaînes d'évènements possibles menant à une situation de défaillance de service ; pour chaque politique partielle, pour chaque état non étendu, il est donc possible d'obtenir une estimation de la probabilité de ne pas respecter la contrainte PCTL en effectuant la somme du produit des coupes minimales actualisées, c'est-à-dire des coupes minimales pour lesquels les évènements s'étant déjà produits ont une probabilité de 1.

Cependant, ce calcul ne donne pas nécessairement une heuristique admissible : les coupes minimales obtenues contiennent à la fois des transitions exogènes et des transitions contrôlables, et pour que l'heuristique soit admissible il est nécessaire de fixer les probabilités des transitions contrôlables ne s'étant pas produites à 0. Avec cette règle, l'heuristique est bien admissible puisqu'elle est optimiste : elle autorise plus d'états et d'actions que nécessaire, par exemple s'il existe dans un état deux actions permettant chacune de valider une contrainte PCTL mais qu'aucune action unique ne permette de les valider toutes les deux.

Le fait de fixer la probabilité des évènements contrôlables à 0 est néanmoins une hypothèse faisant perdre beaucoup d'information à l'heuristique, et en fonction du problème on pourra être amené à choisir une hypothèse plus souple, bien que non admissible ; cependant, il est intéressant de noter qu'il n'est pas indispensable que l'heuristique d'atteignabilité soit particulièrement précise, puisque dans le cas du modèle GMD les objectifs de sécurité et d'optimalité coïncident le plus souvent : en cherchant à remettre le système dans le meilleur état possible, on participe dans la plupart des cas à le rendre aussi résilient que possible aux défaillances futures. Rappelons par ailleurs que le calcul des probabilités des contraintes PCTL doit être effectué sur les probabilités de défaillance dans un faible incrément de temps, et non sur les probabilités de transition du PCMDP qui sont des ratios entre plusieurs probabilités de défaillance.

Enfin, puisque toutes les contraintes PCTL sont des contraintes d'évitement, c'est-à-dire qu'elles spécifient qu'une certaine situation redoutée doit être évitée avec une probabilité minimale, alors il est possible d'oublier les contraintes dans l'exploration après avoir passé un certain nombre d'évènements : si une politique partielle donnée a déjà réalisé l'exploration de tous les états les plus proches de l'état initial et a pu conclure que les actions choisies pour ces états permettaient de garantir que l'état initial respectait une certaine contrainte (la somme des probabilités des évènements découverts menant à des états défaillants ou à des états non étendus est inférieure au seuil p de la contrainte PCTL), alors il est possible d'omettre cette contrainte pour le reste de l'exploration ; de plus, si ceci est le cas de toutes les contraintes, alors il est possible de passer sur une exploration de type LAO*[HZ01] pour compléter le reste de la politique partielle. Ceci implique qu'il n'est nécessaire d'explorer qu'un nombre très faible d'états sous la forme d'un Branch and Bound - ceux qui sont les plus proches de l'état initial.

Dans les paragraphes précédents, nous avons détaillé ce qu'il était possible de faire à partir du modèle GMD, en utilisant des outils de vérification formelle ainsi que l'algorithme Fast-PCMDP ; en particulier, l'exploitation d'un modèle dans le formalisme GMD permet d'évaluer (1) l'impact d'une défaillance en termes de capacité immédiate et de capacité à venir, (2) l'impact d'une défaillance ou d'une reconfiguration sur la résilience du système à des défaillances futures, ainsi (3) qu'une aide à la décision de reconfiguration, soit ponctuelle telle qu'au travers de la génération d'une MEL, soit sur le long terme au travers de la génération de tables de reconfigurations. Dans les paragraphes suivants, nous évaluerons alors dans quelle mesure ces informations participent effectivement à résoudre la problématique que nous avions posée en début de chapitre, c'est-à-dire l'étude des problèmes de décision sur des systèmes critiques de taille industrielle.

VIII.3 Évaluation du processus complet en termes d'ergonomie et de performance

Comme nous l'avons évoqué précédemment, une des limitations principale que nous avons rencontrée lors de notre évaluation concerne le nombre réduit de modèles présents aujourd'hui dans l'industrie qui comportent des notions de modes de fonctionnement ou modes de défaillance : la plupart des modèles disponibles s'intéressent soit au comportement du système, c'est-à-dire une simulation précise des signaux tels que la position ou la loi de commande ; une autre partie des modèles est construite à des fins de diagnostic ou de sécurité, et détaille la propagation d'un signal booléen dans un système ; aucun modèle industriel n'a à notre connaissance été formulé en termes de capacités. Ce constat a deux implications critiques pour notre évaluation :

- Certaines données ne sont pas disponibles pour des systèmes existants, telles que la liste des modes de chaque ressource ou encore les relations logiques entre certaines qualités de service.
- Une majorité des systèmes actuels sont *simples*, en ce qu'ils ne présentent pas ou peu de reconfigurations dynamiques, pas de reconfigurations coordonnées, ainsi que des valeurs de qualités de service binaires ; en particulier, avec de telles simplifications, le besoin industriel actuel en des méthodes de reconfiguration plus avancées que les méthodes existantes est limité. Notons que ce constat ne présage en rien du besoin industriel futur en de telles méthodes.

En gardant à l'esprit cette limitation, nous pouvons procéder de façon similaire à l'évaluation réalisée sur le processus outillé construit pour la résolution du problème du Business Jet : le processus outillé peut être évalué dans un premier temps en termes de performance, en particulier concernant les temps de calcul des algorithmes impliqués dans les différentes étapes du processus, puis dans un second temps en termes d'utilisabilité (c'est-à-dire dans quelle mesure le processus outillé est pratique à utiliser) et de faisabilité (c'est-à-dire dans quelle mesure il est facile d'obtenir les informations et les équipements nécessaires au processus).

VIII.3.1 Évaluation des performances de la résolution en temps de calcul

Sur l'architecture de la chaîne de données RNP-AR, nous avons pu évaluer en conditions réalistes les algorithmes de propagation de la qualité de service, d'évaluation de contraintes de sécurité au travers d'un calcul de coupes minimales ainsi que de calcul de la Minimum Equipment List. Les temps que nous évoquons ici sont à mettre en relation avec les ordres de grandeur suivants :

- Le système complet est décrit en 653 lignes dans le DSL et 988 lignes en Altarica.
- Il est composé de 35 ressources et de 113 services.
- 4 services sont marqués pour l'observation, c'est-à-dire que la perte de ces services est considérée critique. Il s'agit des services de sortie pour les deux chaînes parallèles (gauche et droite).

Les algorithmes ont été réalisés dans le langage Java 1.6, en dehors de l'algorithme pour la génération des coupes minimales, et les expériences ont été menées sur un ordinateur possédant un processeur double cœur à 2.4GHz et 8 Go de RAM, représentatif du type d'ordinateurs de bureau qui peut être mis à disposition d'un ingénieur de conception en architecture.

Temps de résolution pour les différents algorithmes

Comme nous l'avons déjà évoqué précédemment, la mise à jour des qualités de service du système met moins de **5 milli-secondes**, avec une moyenne à 1.2 milli-secondes ; elle nécessite en moyenne 2 passes de l'algorithme de mise à jour sur le système. Cette opération est réalisée une seule fois lorsqu'il s'agit d'évaluer les conséquences d'une défaillance ; elle est réalisée à chaque étape du parcours en profondeur des algorithmes de MEL et de génération des tables de reconfiguration ; elle n'est en revanche pas utilisée par le calcul des coupes minimales, qui s'appuie sur un outil extérieur réalisant l'exploration du système dans le formalisme Altarica. Nous pouvons conclure de ces valeurs que l'algorithme de mise à jour semble particulièrement efficace, bien qu'il puisse être amélioré (de façon marginale) à la fois en temps et en mémoire utilisés si on utilise une implémentation des algorithmes dans un langage permettant un contrôle plus fin sur la mémoire tels que C++ : il est vraisemblable que de meilleures performances soient difficiles à atteindre, hormis éventuellement en parallélisant les mises à jour des

services de types similaires (mise à jour en parallèle des services entrants, synchronisation, puis mise à jour en parallèle des services sortants), ce qui nécessiterait une gestion particulière de la mémoire pour permettre ce type d'écriture de données.

L'algorithme de calcul des coupes minimales se déroule en 0.15 **secondes**, ce qui comprend la traduction du modèle dans le langage Altarica, la résolution par un outil extérieur en fixant 4 observateurs dans le modèle, puis la récupération des résultats. Il est clair qu'il s'agit d'un temps négligeable dans un processus de conception.

L'algorithme de calcul des conditions MEL se déroule en 900.5 **seconde** (soit 15 minutes et 0.5 secondes), avec l'option d'une hypothèse pessimiste, comprenant une exploration en profondeur de 2872 configurations, réparties sur 9 étages - ce qui signifie que certaines configurations étudiées n'étaient atteintes que dans des situations très rares où 8 composants de la chaîne ne sont pas en mode nominal ; notons que ce nombre élevé de configurations possibles découle du fait que tous les changements de modes ne sont pas des défaillances, puisque le modèle prend par exemple en compte une logique de mise à jour des plans de vol, pouvant potentiellement causer la propagation de données erronées. À titre de comparaison, le nombre de configurations marquées comme autorisées pour le décollage est de 355, dont certaines contiennent 6 composants défaillants. Nous avons réalisé cette résolution au travers d'un algorithme simple implémenté en Java, et il est certain que la connexion avec des algorithmes plus performants tels que celui présenté dans [TKIS11] permettrait d'obtenir des performances bien meilleures ; cependant, pour les besoins des modèles que nous avons rencontrés ce type de performance est bien suffisant. Notons néanmoins que sur les 900.5 secondes de résolution, on peut estimer que 43 secondes sont consacrées à la mise à jour du modèle, ce qui représente un temps difficile à réduire.

Enfin, l'algorithme de calcul des tables de reconfiguration n'a pas pu être évalué sur le modèle d'origine, qui ne comporte aucune reconfiguration dynamique. Nous pouvons expérimenter que la génération de tables de reconfiguration est réalisable, en créant artificiellement des modes de fonctionnement pour certains composants, mais le seul apport de cette expérience est de confirmer que l'algorithme Fast-PCMDP est correct - ce qui a déjà été évalué dans le chapitre précédent. Lors de nos tests, les temps de résolution se sont avérés similaires aux temps de calcul des conditions MEL, bien que ces temps puissent varier de façon drastique selon le type d'ajouts que nous réalisons au modèle : en ajoutant artificiellement certaines actions de reconfiguration, certaines contraintes peuvent être violées plus rapidement - donnant un temps de résolution très court - ou au contraire certaines configurations qui n'étaient pas possibles avant peuvent désormais l'être - ce qui donne un temps de résolution plus élevé. Nous pouvons par ailleurs conclure que le choix de la génération d'une table - par opposition à un calcul en temps réel à bord de l'avion - est ici adapté : bien qu'une résolution de l'ordre de quelques minutes soit réalisable *en ligne*, c'est-à-dire par des calculateurs embarqués dans l'avion, ce temps de résolution est suffisamment long pour être problématique, par exemple vis-à-vis de la certification. Puisque notre étude est réalisée dans un cadre industriel, ce type de considération doit être étudié ; la réponse que nous pouvons apporter face à ce problème d'implémentation est donc la génération au sol d'une table de reconfiguration, associée à un système embarqué dans l'avion qui exploite cette table, au travers d'une connexion avec les systèmes de diagnostic embarqués pour identifier la configuration courante. Notons enfin que nous n'avons pas essayé dans nos tests d'optimiser la résolution, par exemple en utilisant certaines des techniques de calcul d'heuristique que nous avons évoqué précédemment, notre objectif n'étant pas la maximisation de la performance mais l'étude de faisabilité.

Évaluation du passage à l'échelle industrielle par l'étude du dimensionnement des variables

Ces valeurs ne constituent bien sûr qu'un exemple isolé, ne permettant pas de généraliser dans quelle mesure ces algorithmes de résolution permettent de passer à l'échelle, c'est-à-dire de traiter des modèles dont la taille est potentiellement plus grande de plusieurs ordres de grandeur ; les expériences que nous avons réalisées précédemment, en particulier concernant l'algorithme PCMDP, nous permettent néanmoins d'être optimiste : dès lors que nous disposons d'heuristiques suffisamment informatives, telles que pour le business jet ou pour le domaine GridWorld, le temps de résolution demeure tout à fait réaliste pour des espaces ayant un nombre d'états de l'ordre de 10^5. Nous pouvons alors prédire

un ordre de grandeur des performances en réfléchissant en termes de dimensionnement des variables.

En termes de PCMDP, les deux variables importantes pour le dimensionnement sont le nombre d'états N_s et nombre d'actions N_a : en effet, le nombre de politiques déterministes qu'il est potentiellement nécessaire d'explorer est $N_a^{N_s}$, correspondant au fait d'associer une action à chaque état. Le facteur critique est donc le nombre d'états, puisqu'il fait augmenter l'espace de recherche de façon exponentielle.

En termes de formalisme GMD, le nombre d'actions découle directement du nombre de transitions contrôlables. Il s'agit donc du nombre de reconfigurations possibles, dès lors qu'on utilise un algorithme éliminant les reconfigurations absurdes ; pour des systèmes réels, nous pouvons estimer ce nombre à un ordre de grandeur de 10 pour une majorité de problèmes d'étude et un maximum de 100 pour des systèmes comportant beaucoup de sélecteurs, tels que des réseaux électriques. Par conséquent, dans le dimensionnement, nous devrons multiplier le nombre d'états estimés par 2 pour prendre en compte le nombre d'actions.

Concernant le nombre d'états, il correspond directement au nombre de configurations possibles, c'est-à-dire au produit du nombre de modes pour chaque ressource. Une majorité de ressources ne sera décrite qu'avec quelques modes de défaillances possibles et quelques modes de fonctionnement ; dans les essais que nous avons menés, le nombre maximal de modes que nous avons été amenés à fixer était de 5 ; nous pouvons donc évaluer 5 comme un maximum pour le nombre de modes en première approximation ; rappelons que les modes correspondent à l'état interne d'une ressource et non à l'état de ses services, et qu'il n'est intéressant de distinguer deux modes que s'ils ont des effets différents en termes de qualité de service sortant. Si on fixe un nombre de systèmes N alors le nombre d'états (au sens PCMDP) obtenu est de l'ordre de $2 * 5^N$.

Si on compare ces résultats avec ceux obtenus lors de l'évaluation de l'algorithme Fast-PCMDP, nous pouvons en conclure que $N = 7$ donne des systèmes pouvant être résolus à un horizon de l'ordre de la minute et $N = 11$ est un maximum, à la fois en termes de temps (de l'ordre de la journée) et en termes de mémoire utilisée sur un ordinateur classique. Il est important de noter que ceci ne concerne que les ressources ayant des modes de défaillances, puisque l'ajout de ressources n'ayant pas de modes de défaillance (tels que des fonctions ne pouvant pas défaillir, ou des équipements dont on ne souhaite pas considérer la défaillance tels que des câbles) n'impacte pas le nombre d'états différents - par exemple dans le cas du modèle de la chaîne de données que nous avons capturé, qui comporte 35 ressources mais ne donne lieu qu'à une exploration de 2872 états. Néanmoins, ce nombre impacte le temps de mise à jour des qualités de service, qui devient non négligeable sur des états aussi grands ; il est donc vraisemblable que $N = 11$ soit en réalité une valeur optimiste, puisqu'il faut y ajouter un ensemble d'opérations coûteuses telles que la mise à jour des qualités de service.

Ce que nous pouvons conclure de cette étude de dimensionnement est que ces méthodes sont adaptées pour des études de sous-ensembles de systèmes ou des études amonts, c'est-à-dire lorsque l'objectif est de regarder le comportement d'un nombre réduit d'aspects du système ; par exemple, sur un même système il est possible de réaliser une première étude centrée sur les données fonctionnelles telles que la précision de la position ou les dépendances de service de la capacité RNP-AR, puis une seconde étude séparée sur des données physiques telles que les alimentations électriques de certains systèmes ; un modèle cherchant à allier les deux pourrait dépasser la taille maximale qu'il est raisonnable de traiter. Cependant, ceci rejoint la pratique actuelle en termes de modélisation, qui consiste à obtenir plusieurs modèles à des fins de preuves précises, et non un modèle complet et détaillé de l'ensemble du système qui simulerait parfaitement celui-ci.

VIII.3.2 Évaluation de l'utilisabilité du processus outillé

En termes d'utilisabilité, on s'intéresse à la fois à la qualité du résultat obtenu et au fait qu'il remplisse réellement l'objectif fixé, c'est-à-dire qu'il aide à la conception de systèmes sûrs et optimaux. Il est important de noter cependant que nous ne nous intéressons pas, ou peu, au modèle du système en lui-même : dans l'exemple de la chaîne de données de la capacité RNP, l'objectif n'était pas de trouver la meilleure architecture répondant aux contraintes du RNP, mais était d'évaluer si le processus outillé était adapté à la recherche de cette architecture ; c'est pour cette raison que les détails de l'architecture (en particulier les logiques de consolidation de sources) ne sont pas importants pour notre étude.

Concernant la qualité du résultat obtenu, nous avons montré que ce modèle permettait effectivement de représenter les capacités d'un système au travers de la notion de qualité de service, de vérifier le respect de contraintes d'optimalité et de validité, ainsi que de synthétiser des aides à la décision, sous la forme de tables MEL et de tables de reconfiguration mais également sous la forme de propositions d'architectures lorsqu'on couple le formalisme GMD à des algorithmes d'optimisation. En termes de qualité du résultat, l'objectif est donc atteint.

Concernant les aspects d'aide à la décision, nous avons évoqué quelques modifications que nous avons mises en place dans l'outil, par exemple en permettant d'écrire des équations logiques plus complexes entre les entrées et sorties d'une ressource. L'un des exemples flagrant de concept difficile à exprimer dans le formalisme GMD est celui de lien de communication : si on souhaite dire qu'un service passe d'une ressource A à une ressource B par un câble intermédiaire, on souhaiterait intuitivement dire que la valeur de la ressource entrant dans le câble est la même valeur que la ressource sortant dans le câble, sauf si le câble est cassé auquel cas la qualité de service sortante est la plus basse possible ; ceci n'est pas exprimable dans le formalisme GMD puisque pour un mode donné une seule qualité de service en sortie est permise (en plus de la qualité minimale si la ressource est inconsistante). En vérité, pour exprimer ce type de relation dans le formalisme GMD il serait nécessaire de dire que le câble fournit un service de communication, pouvant être défaillant, et que la ressource B conditionne la réception des services entrants sur le fait que le service de communication est fonctionnel ; ce type de considération logique, bien que correct, n'est manifestement pas intuitif ou lisible pour les utilisateurs actuels, qui sont habitués à un autre schéma de pensées.

Ces modifications montrent que l'utilisateur a besoin de pouvoir exprimer des schémas qui ne sont pas disponibles nativement dans le formalisme GMD, mais qui peuvent être rendus disponibles au travers d'un DSL. Ceci est un point positif pour notre processus outillé, puisque nous avons pu surmonter cette difficulté en créant un Domain Specific Language adapté, mais un point négatif pour le formalisme GMD qui peut sembler trop réduit par rapport aux besoins de l'utilisateur.

Néanmoins, ce choix de restreindre le formalisme GMD à un minimum cohérent est parfaitement assumé : en ayant choisi un périmètre très réduit pour l'expressivité du formalisme, nous avons pu montrer des résultats particulièrement forts sur la convergence de la mise à jour et l'unicité de l'état ; ce type de résultats ne peut pas toujours être obtenu avec des formalismes plus expressifs, comme nous en avons déjà montré un exemple précédemment. Pour tirer partie du meilleur des deux mondes, entre l'expressivité et la possibilité de preuves de convergence, la solution que nous avons choisie est de laisser à la charge de l'outil de rapprocher le DSL et le formalisme GMD : en effectuant des vérifications supplémentaires sur le modèle plus expressif saisi dans l'outil, par exemple sur des propriétés de monotonie de la qualité de service, nous pouvons montrer au cas par cas que les modèles saisis rentrent dans le cadre des preuves réalisées sur le formalisme GMD.

Enfin, les considérations d'ergonomie que nous avons évoquées en début de ce chapitre sont essentielles pour l'utilisabilité du processus outillé : bien que nous ayons réduit les concepts du formalisme GMD au minimum, nous manipulons des techniques mathématiques avancées telles que les Logiques Temporelles, la synthèse de politique pour des processus décisionnels markoviens, ou même la notion de qualité de service ; une majorité des utilisateurs à qui est destiné le processus outillé ne sont pas familiers de ces concepts, et l'acceptance du processus est donc conditionnée au fait qu'il puisse véhiculer clairement et intuitivement des informations complexes, par exemple sur ce que doit faire l'utilisateur pour capturer un système dans l'outil ou sur ce que signifient les résultats obtenus par les algorithmes. Nous n'avons pas réalisé d'études spécifiques pour évaluer l'ergonomie de notre processus outillé, bien que de telles études soient possibles avec des méthodes scientifiques, mais le retour d'expérience que nous pouvons proposer sur ces considérations est que l'ensemble des améliorations d'ergonomie que nous avons détaillées participe de façon positive au confort de l'utilisateur.

VIII.3.3 Faisabilité du processus outillé

Enfin, concernant la faisabilité, il est possible de considérer les matériaux et informations nécessaires à la mise en œuvre du processus outillé : comme nous l'avons déjà évoqué précédemment, l'un des freins à la mise en œuvre de certaines des techniques que nous proposons, en particulier concernant les tables de reconfiguration, est qu'il est nécessaire de disposer d'équipements spécifiques pour capturer la

qualité de certains services à des points clés du système ; ces équipements peuvent être conçus de façon similaire aux équipements de surveillance de type BITE (Built-In Test Equipment), voire directement en se basant sur de tels équipements ; la difficulté est alors de pouvoir coordonner plusieurs fournisseurs d'équipements différents afin que chaque équipement envoie des informations cohérentes sur l'état de ses services et des services entrants ; ceci n'est pas un problème insurmontable, puisque ce type de réflexion est déjà nécessaire pour les domaines de la surveillance (Flight-Warning) et de la maintenance, mais ce problème nécessite néanmoins un effort particulier, ce qui implique que l'utilisation des méthodes de reconfiguration doit pouvoir présenter une valeur ajoutée nette, en particulier en termes financier, pour justifier le coût des changements de procédures. Notons cependant qu'en utilisant les systèmes de surveillance déjà en place, il ne sera vraisemblablement pas nécessaire d'ajouter de nouveaux équipements, mais simplement de modifier le type de messages envoyés par la surveillance.

Le deuxième point potentiellement bloquant en termes d'équipement concerne l'affichage des informations de capacité au pilote ; plusieurs visions sont possibles dans ce domaine, dont certaines sont protégées par des brevets, si bien qu'une étude particulière est nécessaire pour concevoir un équipement ou une interface adapté à l'interaction avec le pilote sur des questions de qualité de service. L'étape suivante serait l'automatisation d'une partie des reconfigurations optimales, mais qui est problématique pour les équipements actuels puisque toutes les fonctionnalités de l'avion ne sont pas automatisables. En raison de tout ces paramètres inconnus, il n'est pas possible de conclure à l'heure actuelle sur la faisabilité industrielle d'un système d'aide à la décision pour le pilote orienté sur la gestion des capacités, ou tout du moins sur la rentabilité d'une telle solution.

Concernant l'aide à la décision lors des phases de conception, la conclusion est plus simple : le processus ne nécessite qu'un ordinateur classique, et les données ne nécessitent qu'un expert du domaine, ce que possèdent déjà les entreprises étant amenées à être intéressées par ce type d'outils. Contrairement au problème d'obtention des données que nous avions rencontré pour le problème du Business Jet, les données que nous manipulons avec ce processus outillé sont des données qui sont déjà collectées et analysées à l'heure actuelle : le processus outillé ne fait qu'assister un concepteur dans une tâche qu'il réalise déjà actuellement par d'autres moyens. La nuance sur ce point concerne les données nécessaires à la reconfiguration : ces données ne sont pas disponibles, mais parce qu'aucun système actuel n'a, à notre connaissance, de problématique de reconfiguration coordonnée (au sens de la reconfiguration de modes de fonctionnement). Il ne s'agit donc pas d'un problème d'obtention de données, mais d'un problème de besoin : dès lors qu'il existera un besoin industriel de systèmes reconfigurables, les évaluations que nous avons menées montrent qu'il n'y aura vraisemblablement pas de frein à leur conception en termes de faisabilité, à la fois sur l'obtention des données et sur la puissance de calcul requise.

VIII.3.4 Conclusion et résumé des apports sur l'évaluation du processus outillé basé sur le formalisme GMD

Dans les chapitres précédents, nous avions identifié que plusieurs problèmes de décision dans le contexte avionique pouvaient être exprimés en termes de gestion de la qualité de service d'un système ; nous avions alors construit un formalisme, le modèle de Gestion de Modes Dégradés, centré sur la notion de qualité de service et sur le fait que celle-ci doit respecter des contraintes de sécurité et d'optimalité. Ceci nous avait amenés dans un second temps à concevoir un algorithme de résolution adapté, permettant la production de stratégies de reconfiguration - pour un système exprimé dans le formalisme GMD - qui respectent les contraintes d'optimalité et de sécurité.

Dans ce chapitre, l'enjeu était alors de construire un processus outillé complet autour de ce formalisme et de cet algorithme, puis d'évaluer dans quelle mesure ce processus permet d'aider à la prise de décision lors des phases de conception et lors des phases opérationnelles. Nous avons réalisé ceci, de façon plus spécifique, au travers des apports suivants :

- Nous avons établi que des **principes d'ergonomie** tels que l'implémentation d'un DSL ou le choix d'un affichage graphique automatique contribuaient à la facilité de capture du modèle, et donc à sa qualité finale.
- Nous avons montré que le formalisme GMD permettait effectivement la **capture de systèmes industriels**, au travers de la définition et de l'étude d'un modèle décrivant la chaîne de données impliquée dans la capacité RNP-AR d'atterrissage à haute précision.

- Nous avons mis en œuvre les algorithmes de mise à jour et de **recherche de coupes minimales**, qui permettent une évaluation de l'impact des défaillances en termes de capacité (qualité de service) et de sécurité.
- Nous avons prouvé que la **génération automatique de tables d'aide à la décision** était possible, en particulier concernant les conditions MEL et les décisions de reconfiguration.
- Nous avons **évalué le formalisme GMD** et montré que les performances des algorithmes, leur faisabilité et l'utilisabilité du processus complet étaient satisfaisantes dans des cas d'utilisation classiques.

Néanmoins, cette évaluation a aussi mis en avant trois limitations principales : la première est que les techniques avancées de reconfiguration de fonction ne répondent pas à un besoin industriel actuel ; nous avons montré qu'elles étaient réalisables, et développé un algorithme et un formalisme permettant de les utiliser, mais la plus-value de ces techniques dans les produits actuels n'est pas apparente, comparée au coût en changement de processus et en équipement qui serait nécessaire pour mettre en œuvre des stratégies de reconfiguration dynamique.

La seconde limitation est que le formalisme GMD est trop restreint, en particulier par rapport aux habitudes de modélisation des utilisateurs ; ceci est compensé par le fait qu'une partie des restrictions peut être levée par l'outil, par exemple au travers d'un DSL, si bien que cette limitation montre simplement que le formalisme GMD ne constitue pas un langage utilisable, et donc qu'il est préférable de ne pas demander à un utilisateur d'exprimer directement des modèles GMD.

Enfin, la troisième limitation concerne le passage à l'échelle : nous avons évoqué certaines optimisation qu'il était possible de réaliser sur les différents algorithmes que nous avons utilisés dans le processus outillé, par exemple en exploitant la forme particulière du modèle ou encore au travers de la parallélisation des algorithmes. Dans les exemples que nous avons rencontrés, ce besoin de passage à l'échelle n'était pas présent, en particulier parce que les décisions auxquelles nous nous intéressons (décision de conception, choix des reconfigurations...) ne s'intéressent qu'à des sous-ensembles du système complet. Cependant, il est évident que la question de l'augmentation de la taille des modèles n'est pas résolue dans notre étude, à la fois en termes de temps de calcul des algorithmes sur ces modèles mais aussi en termes de capture et de gestion des modèles saisis.

ANALYSE CRITIQUE DE L'ÉTUDE

Quel est au final l'apport de cette étude ? L'objectif dans ce court chapitre est de répondre à cette question de deux façons : d'une part, nous souhaitons revenir sur les résultats principaux de nos travaux, en mettant en valeur les avantages et les limites des solutions que nous apportons aux problèmes rencontrés, et d'autre part nous pouvons effectuer un retour sur l'état de l'art actualisé, c'est-à-dire préciser le positionnement de nos résultats vis-à-vis des travaux existants avant et pendant la durée de la thèse.

Modèle PCMDP

Le premier résultat que nous pouvons mettre en avant est algorithmique : cette étude nous a amené à étudier en détail les problèmes alliant les processus décisionnels markoviens et les contraintes de Logique Temporelle PCTL, et plus particulièrement lorsque l'alliance est réalisée sous la forme d'un problème PCMDP ; le modèle PCMDP étant récent dans la littérature, nous avons réalisé de nombreux apports sur la compréhension du modèle et de ses limites, sa complexité, la forme de ses solutions et enfin sur des méthodes de résolution - différentes selon les hypothèses simplificatrices choisies.

Nous pouvons mettre en avant l'apport principal du modèle PCMDP, qui est la garantie forte de respect du type de contraintes de sécurité qui est utilisé dans l'industrie pour les analyses de sûreté de fonctionnement. En ce sens, le modèle PCMDP et les algorithmes de résolution que nous proposons sont une réussite puisqu'ils permettent effectivement de concevoir des contrôleurs sûrs et optimaux pour des systèmes critiques. Nous avions détaillé dans l'état de l'art les apports du modèle PCMDP vis-à-vis d'autres études, qui sont à présent renforcés par le fait que des méthodes de résolution plus efficaces existent.

Cependant, non seulement les résultats que nous avons obtenus l'ont été en se basant sur une simplification du modèle PCMDP - simplification des contraintes pour SPC MDP et simplification du type de politique cherché pour Fast-PCMDP - mais les performances en termes de temps de calcul demeurent problématiques : dans les cas d'applications que nous avons étudiés, ces performances semblent suffisantes, mais PCMDP demeure un problème beaucoup plus complexe que d'autres approches. Nous avons pu voir dans notre étude que cette réflexion était par exemple rencontrée pour le cas du Business Jet, pour lequel il est difficile d'argumenter en faveur d'un modèle PCMDP compte tenu du fait que la représentation sous la forme d'un problème d'optimisation sous contraintes est à la fois plus simple et plus répandue dans l'industrie.

Notre étude a donc mis en avant deux limites au modèles PCMDP : (1) sa complexité en termes de temps de résolution, qui limite la taille des modèles pouvant être traités, et (2) le manque de modèles industriels ayant des problématiques de type PCMDP, impliquant qu'il est difficile d'évaluer quels travaux seraient les plus intéressants pour faire évoluer le modèle PCMDP. Il est en effet possible de prévoir plusieurs perspectives d'évolution du modèle, en particulier des évolutions vers l'observabilité partielle ou vers des processus décisionnels markoviens à temps continu, mais l'intérêt de ces évolutions est discutable tant qu'une problématique industrielle ne justifie pas de leur nécessité.

À l'inverse, d'autres hypothèses simplificatrices que celles que nous avons choisies permettent des résultats intéressants, comme les travaux de [LPH15] qui se concentrent sur la synthèse de contrôleur respectant des contraintes LTL, dans un contexte où l'objectif ne peut pas être rempli avec une probabilité 1. Le choix d'une simplification dépend bien évidemment du cas d'application, mais la tendance que nous pouvons observer pour les perspectives futures est que le problème PCMDP définit un cadre global complet mais complexe, qui gagne à être adapté aux besoins d'un cas d'application spécifique pour améliorer les performances de résolution.

En particulier, la performance de notre algorithme Fast-PCMDP est conditionnée par la qualité des heuristiques : nous avons proposé une méthode générale permettant d'obtenir les heuristiques nécessaires sous condition que le problème soit exprimé sous une forme STRIPS, mais nous avons aussi montré que dans le cas d'application du modèle GMD il était intéressant de construire nous-même des heuristiques adaptées, qui étaient plus proches du problème qu'une heuristique générale. Des travaux intéressants [PZ14] ont cependant été réalisés par la communauté de la planification sur d'autres méthodes de simplification de l'espace d'états et l'espace d'actions, qui laissent penser qu'une heuristique générale plus performante pourrait être construite - bien que l'impact en termes de performance reste à évaluer. D'autres méthodes utilisent des combinaisons d'heuristiques peu informatives pour obtenir plus d'informations [PNAL15] - qui semblent applicables à notre algorithme puisque notre méthode de recherche est proche des méthodes classiques de type A*.

Enfin, notre algorithme Fast-PCMDP propose un cadre de travail sur lequel il est possible de construire de nombreux algorithmes inspirés de méthodes de recherche heuristique ; il est alors possible d'utiliser - de façon similaire à ce que nous avons proposé - l'immense quantité de travaux existants sur la recherche heuristique au service de la résolution d'un problème PCMDP. L'autre extension possible de l'algorithme Fast-PCMDP est à l'inverse de regarder ce que la méthode que nous avons développée peut apporter à d'autres problèmes du domaine, par exemple la synthèse LTL ou PCTL qui peut être envisagée sous l'angle d'un problème de planification d'un espace immense [DGV15].

Problème du Business Jet

Le second résultat principal de cette étude est la définition et l'étude du problème du Business Jet. L'apport de nos travaux réside à la fois dans la définition des données et de l'enjeu du problème et dans l'étude de faisabilité d'une aide à la décision - qui montre qu'une aide à la décision est réalisable voire nécessaire. Cette étude a de plus mis en avant les limites de ce problème, en particulier concernant l'obtention des données d'entrée.

Nous avons détaillé d'une part les difficultés liées à l'obtention d'une information de coût précise, qui influe directement sur la qualité de la solution. Un des domaines d'étude qui est amené à traiter ce type de questions est celui de la maintenance prédictive, en particulier en se basant sur des techniques d'apprentissage [AOK14] pour anticiper une défaillance à partir de retours d'expérience, ou d'une façon générale en utilisant des techniques d'analyse de données (Data-Mining) [CML14].

D'autre part, nous avons mis en avant le manque de formalisation des données de type MEL ou TSM : ces documents demeurent encore aujourd'hui dans une logique de format "papier" (ou pdf), c'est-à-dire sans une formalisation des données lisibles et interprétables à partir d'un ordinateur. Cette logique est dommageable pour le futur de l'industrie, puisqu'elle limite les possibilités de connexion de ces données avec d'autres systèmes, par exemple pour des équipements d'évaluation de MEL automatiques ou de procédure automatique. Nous avons proposé dans cette étude plusieurs considérations qui nous semblent importantes, mais de tels "Manifestes pour la formalisation des documents" n'ont de sens que dans le contexte de la création d'un nouveau modèle d'avion et de l'amélioration des processus avionneurs - puisque tous les processus d'opérations actuels ont été construits autour du format des documents existants de MEL et TSM.

Enfin, un des autres domaines d'expansion pour le problème du Business Jet concerne la gestion de flotte : lorsqu'on considère une flotte d'avions au complet, il est possible de trouver des stratégies qui minimisent le coût global au niveau de la flotte mais peuvent sembler contre-intuitives à l'échelle d'un avion, par exemple de toujours faire voler en priorité l'avion le plus proche de sa date de maintenance

programmée [Baz15] ; cette vision est intéressante puisqu'une partie des situations rendant difficile la résolution d'un problème du type Business Jet concerne les combinaisons de pannes bloquant l'avion au sol - ce qui peut être compensé au niveau de la flotte si la compagnie aérienne peut disposer d'un avion de remplacement.

Modèle centré sur la qualité de service

Le troisième résultat principal de notre étude est le modèle de Gestion de Modes Dégradés, centré sur la notion de qualité de service. Ce modèle constitue un apport innovant vis-à-vis de la littérature existante pour deux raisons : (1) il prend en compte une forme de contrat de service entre les différents composants de systèmes ; ce type de notion est utilisé dans plusieurs langages informatiques, tels que Ada comme nous l'avons déjà évoqué, mais n'est pas ou peu utilisé au sein des langages de modélisation ; en particulier, le fait qu'il soit possible d'exprimer une dynamique d'un système autour des contrats repose sur (2) la nécessité d'un ordre dans les valeurs de qualité de service, qui diffère des choix de modélisation habituels, centrés sur des domaines (Vrai, Faux) ou (OK, Erroné, KO).

Il est cependant important de noter, comme nous l'avons déjà souligné, que le modèle GMD ne constitue pas en lui même un langage - et n'est d'ailleurs pas adapté pour être un langage qu'un utilisateur devrait directement utiliser. La vocation de ce modèle est de présenter des concepts innovants, au travers d'une nouvelle manière d'envisager la propagation des changements de capacité dans un système, mais il peut être porté par des langages très différents, par exemple des langages tels que Altarica ou le DSL développé au sein de Thales Avionics que nous avons utilisé pour nos exemples. L'une des extensions possible de ce modèle est ainsi de définir un langage spécifique pour la saisie de problèmes, mais une extension - plus intéressante à notre avis - est d'étudier dans quelle mesure les langages existants et les modèles existants peuvent être enrichis avec les concepts de qualité de service ordonnée et de contrat.

Enfin, nous avons abordé à plusieurs reprises la question de l'automatisation partielle ou totale de certaines tâches dans les systèmes avioniques, en particulier puisque le modèle GMD a été défini en prenant en compte les possibilités de reconfiguration dynamique. Notre étude a mis en avant une limite à la notion d'avions reconfigurables dynamiquement, en ce que les problématiques rencontrées dans les différents cas industriels que nous avons pu observer relevaient le plus souvent de problèmes de gestion de la complexité, en particulier sur la représentation et l'analyse d'un système. À notre connaissance - basée sur la vision des projets publics existants - les problèmes de continuité de service suite à une défaillance sont plutôt résolus au travers de l'amélioration des équipements individuels (en particulier l'amélioration du processus de conception de ces équipements) plutôt que par des reconfigurations à l'échelle du système. Ceci n'enlève cependant rien à l'intérêt de nos résultats sur la reconfiguration dynamique, qui demeure une option envisageable dans le futur.

Il est difficile de prédire à l'heure actuelle à quel horizon il est réaliste d'imaginer des avions entièrement automatisés ; en dehors de la faisabilité, que nous avons étudiés dans cette étude dans le périmètre que nous nous étions fixés, d'autres arguments permettent de s'interroger sur la valeur ajoutée de l'automatisation totale : les études montrent par exemple que l'opinion publique est sensibilisée à ce type de considérations, mais que les passagers ont une nette préférence pour la configuration actuelle à deux pilotes [MRWO14]. Ce type d'argument, qui induit une certaine résistance à l'introduction de nouvelles technologies de la part de l'industrie, a cependant un double aspect : d'un côté, il est évident que l'introduction d'une technologie doit apporter suffisamment de valeur vis-à-vis des différents acteurs du domaine avionique pour justifier les coûts de tests, d'études et de conception ; c'est par exemple le cas pour les technologies multi-modales à destination du pilote, telles que la commande vocale [SM14], qui demandent un investissement en temps d'étude et en coût de conception non négligeable pour une valeur ajoutée qui est encore incertaine à l'heure actuelle. D'un autre côté, l'attrait des nouvelles technologies est tel que les pilotes et compagnies aériennes ont naturellement tendance à utiliser les outils disponibles dans le commerce tels que des applications sur tablettes, bien que ces outils aient nécessairement une maturité faible en termes de sécurité ou de sûreté ; de nombreux incidents se sont déjà produits avec l'utilisation de tablette, du plus "léger" tel que l'immobilisation de plusieurs avions au sol suite à une erreur dans une application tablette "d'electronic flight bag" (application contenant

des documents à destination du pilote) [Gar], jusqu'à un avion "rapant" le sol lors du décollage suite à une mauvaise saisie d'une information dans une application iPad aidant à la configuration de l'avion : le pilote a pressé une touche 6 au lieu d'une touche 7, entrant un poids pour l'avion de 66.400kg au lieu de 76.400kg [ATS].

Nous pouvons bien sûr faire le lien avec d'autres enjeux, en particulier concernant la cyber-sécurité [Dul15]. Ce que montrent ces exemples est que nous devons mener une réflexion approfondie sur les méthodes de conception, en particulier pour faire face aux cycles d'adoption des nouvelles technologies qui sont plus rapides aujourd'hui qu'ils ne l'étaient lorsque les avions existants ont été conçus.

Nous rejoignons ici des considérations sur le futur des méthodes de Model-Based Safety Assessment ; le constat, que nous avons repris dans nos travaux, est que les méthodes actuelles ne semblent pas adaptées aux systèmes du futur [RBF14], ce qui implique que des changements de modèles *et* de processus sont nécessaires. En particulier, il semble indispensable pour les outils futurs de favoriser l'intégration des processus de Vérification et Validation (notamment concernant la sécurité) avec les autres processus d'ingénierie.

Retour d'expérience sur la conception d'outils

Le quatrième et dernier résultat principal de cette étude consiste en un retour d'expérience sur la conception de deux processus outillés - pour le problème du business Jet et pour la capture et l'analyse de modèles au formalisme GMD. Les apports sont minimes sur le plan théorique, puisqu'ils consistent d'une façon générale à confirmer que des méthodes existantes (principes d'ergonomie, principes de design, algorithmes de dessin automatique, conception de Domain Specific Language, ...) sont des méthodes efficaces et permettant d'excellents résultats.

Le fait d'avoir réalisé l'exercice de la conception d'un outil orienté vers la simplification de l'interaction avec un utilisateur a cependant mis en avant les limites de plusieurs outils existants actuellement dans le commerce, qui n'ont pas été basés sur les différents principes que nous avons choisis d'utiliser. Il est néanmoins important de souligner le fait que la conception d'un outil intuitif n'est pas triviale, en particulier lorsque les concepts manipulés sont complexes. Nous avons fait le choix de la simplification de ces concepts, en nous limitant à un nombre réduit d'objets et d'interactions autorisées, mais ce choix s'est avéré par moment frustrant pour les utilisateurs, qui ont parfois des habitudes de modélisation ancrées ou des attentes qui ne sont pas remplies par une approche simplifiée. Un autre choix aurait pu être la création d'outils de capture proches des considérations de chaque métier, par exemple sous la forme de vues différentes ou de DSLs plus spécialisés, mais avec potentiellement une complexité supplémentaire pour s'assurer que les différentes vues soient cohérentes les unes avec les autres.

CONCLUSION

Bilan

Dans cette étude, nous avons présenté une réflexion de trois ans sur la thématique de l'autonomie pour les systèmes critiques. Au bilan, à la question "comment ajouter de l'intelligence à un système sans compromettre sa sécurité", nous apportons la réponse (courte) qu'il est nécessaire d'appuyer la conception du système sur un processus outillé de modélisation intégrant une vérification formelle du modèle. Il s'agit d'une réponse qui peut sembler évidente, mais qu'il est important de réaffirmer.

La réponse longue, dont nous avons expliqué la démarche dans ce document, a consisté tout d'abord en la définition du cadre de l'étude, celui des systèmes auto-adaptatifs critiques soumis à des évènements redoutés probabilistes, puis en second lieu en une analyse de deux cas d'applications qui a nécessité, pour chacun de ces cas, la mise en place d'un processus outillé. C'est au travers de la construction de ces deux processus outillés que nous avons pu identifier et développer différentes méthodes qui nous semblent applicables à tous les problèmes de décision sûre et optimale.

En particulier, cette étude a permis de réaliser quatre apports principaux, dans le cadre des communautés de planification en environnement probabiliste et de vérification formelle probabiliste :

- Nous avons développé **deux méthodes de résolution pour le modèle PCMDP**, qui est un modèle permettant d'exprimer des problèmes de décision sûre et optimale dans un contexte probabiliste. Nous avons montré que ces méthodes de résolution avaient des performances plus élevées de plusieurs ordres de grandeur que les méthodes existantes pour PCMDP et évalué les avantages et limites des hypothèses simplificatrices que nous avons choisies vis-à-vis de cas industriels.
- Nous avons défini et étudié le **problème du Business Jet**, en choisissant l'approche la plus générale possible (et réalisable) basée sur le choix d'une dynamique probabiliste et de contraintes PCTL simplifiées. Nous avons prouvé la faisabilité d'un processus d'aide à la décision pour ce problème et mis en avant les axes d'amélioration nécessaires pour une mise en œuvre industrielle de ces méthodes.
- Nous avons défini le **modèle de gestion de modes dégradés**, basé sur les notions de qualités de service et de contrat, qui apporte une représentation originale des capacités d'un système, et de l'impact des évènements redoutés et des actions d'un opérateur sur ces capacités. Nous avons en particulier prouvé que ce modèle disposait de propriétés intéressantes en termes de convergence et de résolution et nous avons montré qu'il permettait d'affronter, dans une certaine mesure, la complexité du système.
- Nous avons implémenté deux **démonstrateurs portant les processus outillés**, basés sur la capture de documents (MEL, TSM) et la communication avec un opérateur humain (DSL, principes d'ergonomie), tout en permettant des connexions intéressantes avec d'autres outils au travers de la génération automatique de modèles ; le développement de ces démonstrateurs offre un retour d'expérience sur plusieurs techniques qui peuvent être mises en œuvre lors de la conception d'un processus outillé.

Discussion

Néanmoins, plus que les apports que nous venons de mentionner - qui ont une place bien définie dans l'état de l'art existant, puisqu'ils se positionnent en réponse aux multiples questions que nous avions identifiées au début de cette étude - nous souhaitons souligner ces trois résultats pour lesquels nous pensons que nos travaux offrent une argumentation pertinente :

- La formalisation du modèle de gestion de modes dégradés a mis en avant **l'importance de la notion de qualité de service dans la gestion des systèmes**.
- La modélisation des cas d'application, au travers du processus outillé, a permis de confirmer que la **conception du système et sa validation doivent être effectuées de façon conjointe**, par les mêmes outils, sous risque de se heurter de front à la complexité du système.
- L'étude de la littérature existante, ainsi que la contribution à cet état de l'art sous la forme de deux nouveaux modèles de type PCMDP, a permis de montrer que **le défi principal des futurs produits avioniques n'est pas algorithmique**, mais est un défi de capture de connaissances.

Ces résultats ne font pas partie des apports principaux de la thèse - ils ne permettent pas de savoir comment concevoir un système sûr et optimal - mais ils constituent néanmoins un bilan des concepts que nous voulons promouvoir au travers de nos travaux ; si la lecture de ce document ne doit avoir qu'une seule conséquence, notre souhait est que ce document soit initiateur d'une discussion sur ces trois résultats.

Perspectives

Enfin, au travers des réflexions que nous avons menées dans ce document, nous avons évoqué à plusieurs reprises des questions fondamentales pour l'avenir du domaine avionique. Nous avons évoqué la question de l'automatisation totale ou partielle de l'avion, c'est-à-dire la question de l'avion sans pilote et ses implications vis-à-vis des méthodes de conception. Nous avons évoqué la "sûreté de fonctionnement du futur" au travers de nouveaux concepts tels que la qualité de service ou les contrats qui peuvent apporter une nouvelle vision face à de nouveaux systèmes. Nous avons évoqué d'une façon générale le problème du "legacy", c'est-à-dire de l'extension de nos travaux aux avions existants, en particulier concernant la formalisation des documents légaux qui n'ont pas franchi le pas du numérique.

Ces questions restent ouvertes. Il est certain qu'elles nécessitent à l'heure actuelle des changements de processus, en particulier sur le rapprochement des processus d'ingénierie modèle et de Vérification et Validation, mais elles nécessiteront peut-être dans un avenir proche des technologies nouvelles, telles que les technologies d'intelligence artificielle que nous avons ébauchées ici - et sur lesquelles il est possible de bâtir un immense champ d'étude.

Face à ces défis à venir, il est nécessaire de garder à l'esprit que l'esprit humain peine à visualiser la croissance exponentielle de la technologie. Pour appréhender la vitesse extraordinaire d'adoption des nouvelles technologies, il nous suffit de comparer l'ordinateur de guidage de la navette Apollo [Kid99], en service de 1966 à 1975, et les derniers smart-phones tels que l'iPhone 6 : pour envoyer des hommes sur la lune, il a suffi d'un système traitant 41.6 instructions à la seconde, contre les 3.36 milliards d'instructions que peut traiter le processeur d'un smart-phone moderne ; un iPhone 6 pourrait en théorie guider 120 million de fusées Apollo à la fois. Il s'agit du même type d'écart technologique que nous nous retrouverons à affronter dans les *années* à venir, en particulier face à des problématiques telles que la popularisation des drones, le développement des objets connectés, l'explosion du trafic aérien suite à l'arrivée de nouveaux acteurs dans les pays émergents, ou encore la cyber-sécurité.

Cette croissance exponentielle des performances technologiques, inarrêtable et inflexible, induit une croissance exponentielle de la complexité, qui nous force à remettre en cause la plupart des fondements du domaine avionique. Mais, plutôt que de percevoir ces nouvelles technologies comme autant de dangers, il nous faut aller contre notre nature révérencieuse et saisir dès à présent, avec audace, les opportunités qui nous permettrons de maîtriser la complexité croissante à venir.

Appendices

ANNEXE A

DÉTAILS SUR L'ANALYSE FONCTIONNELLE DES DANGERS

Gravité des évènements redoutés	**Probabilité maximum par heure de vol**
Catastrophique	$P < 10^{-9}$
Dangereux	$P < 10^{-7}$
Majeur	$P < 10^{-5}$
Mineur	Aucun
Sans effet	Aucun

TABLE 6 – Objectifs de sécurité en termes de probabilité d'occurrence [ARP 4754]

Effet sur l'aéronef	Sans effet	Faible réduction des capacités ou des marges de sécurité	Réduction significative des capacités ou des marges de sécurité	Réduction importante des capacités ou des marges de sécurité	Perte
Effet sur passagers	Dérangement	Souffrances physiques	Possibles blessures	" Relativement peu " de blessures graves aux passagers	Plusieurs décès ou blessures graves
Effet sur l'équipage	Sans effet	Augmentation de la charge de travail	Dérangement ou augmentation importante de la charge de travail	Souffrances physiques ou augmentation très importante de la charge de travail	Décès ou immobilisation
Classification	**Sans effet**	**Mineur**	**Majeur**	**Dangereux**	**Catastrophique**

TABLE 7 – Critère de classification de la gravité des conditions de défaillance (extrait de [Ade11])

ANNEXE B

DESCRIPTION UML COMPLÈTE DU SCÉNARIO BUSINESS JET

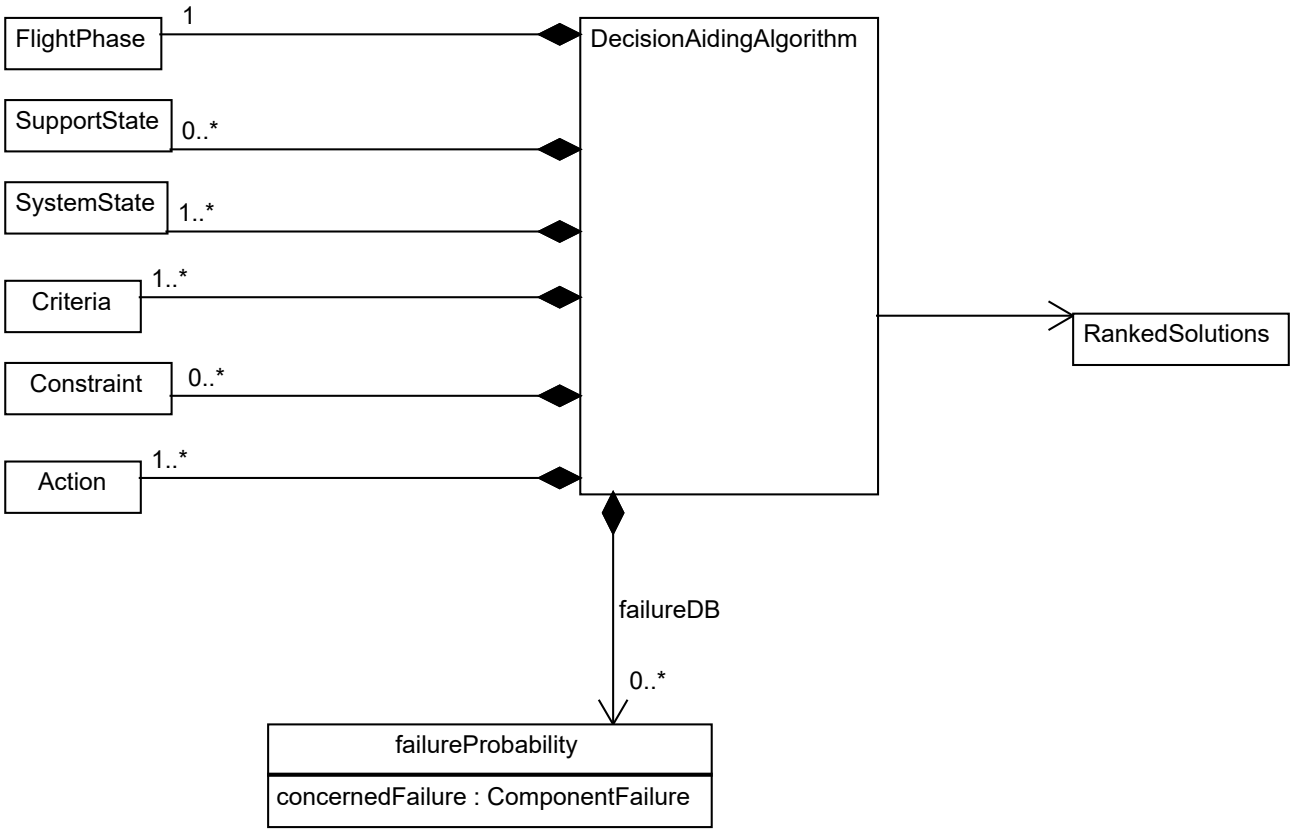
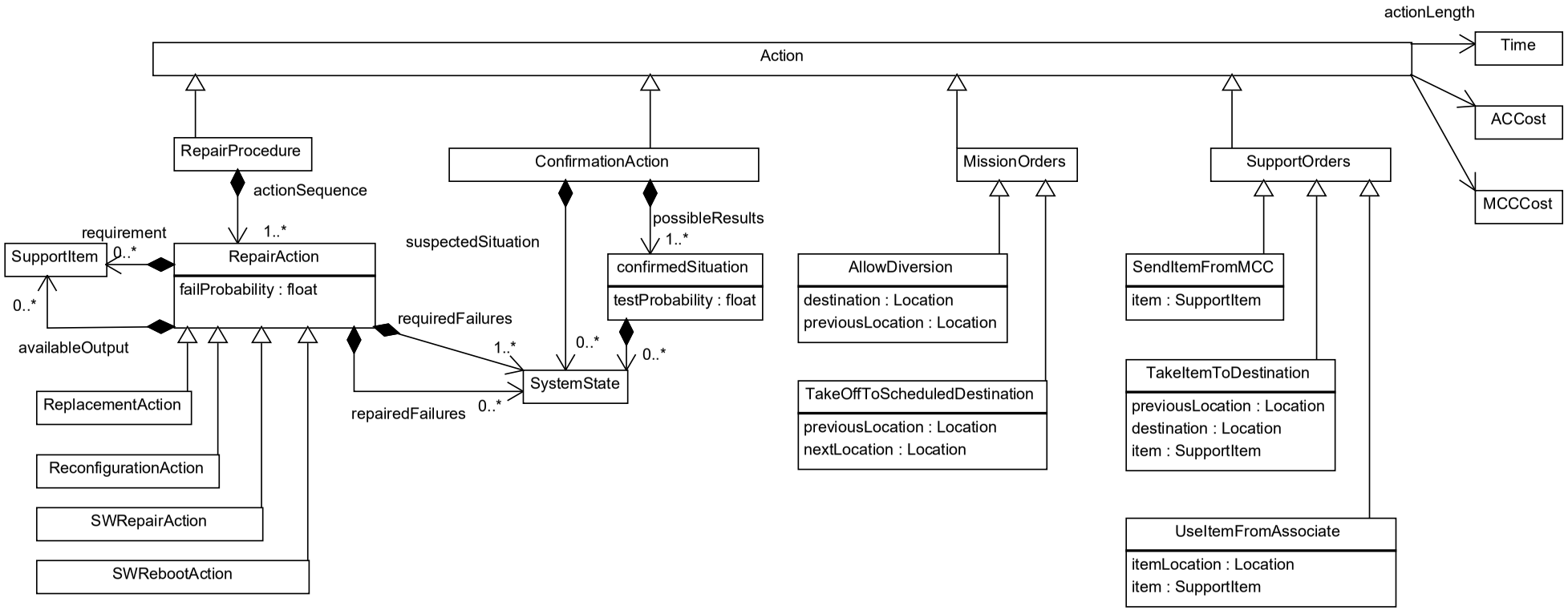
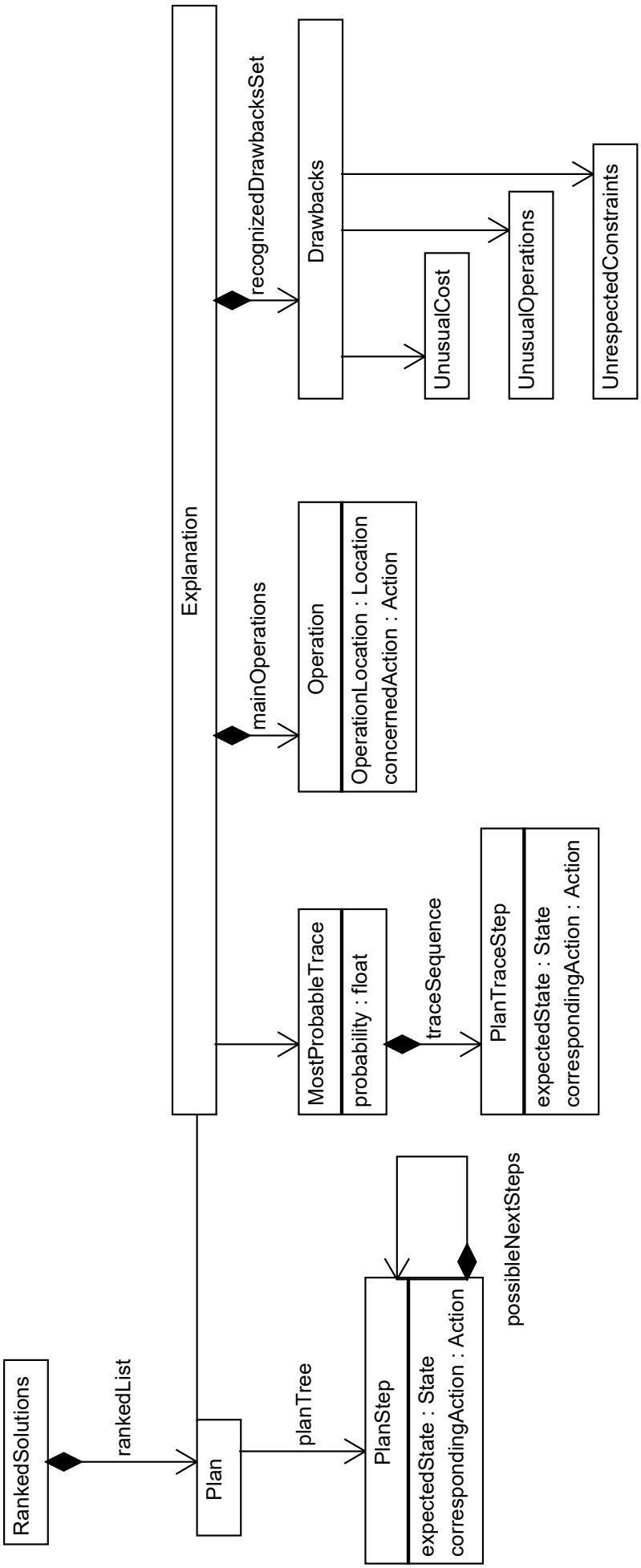
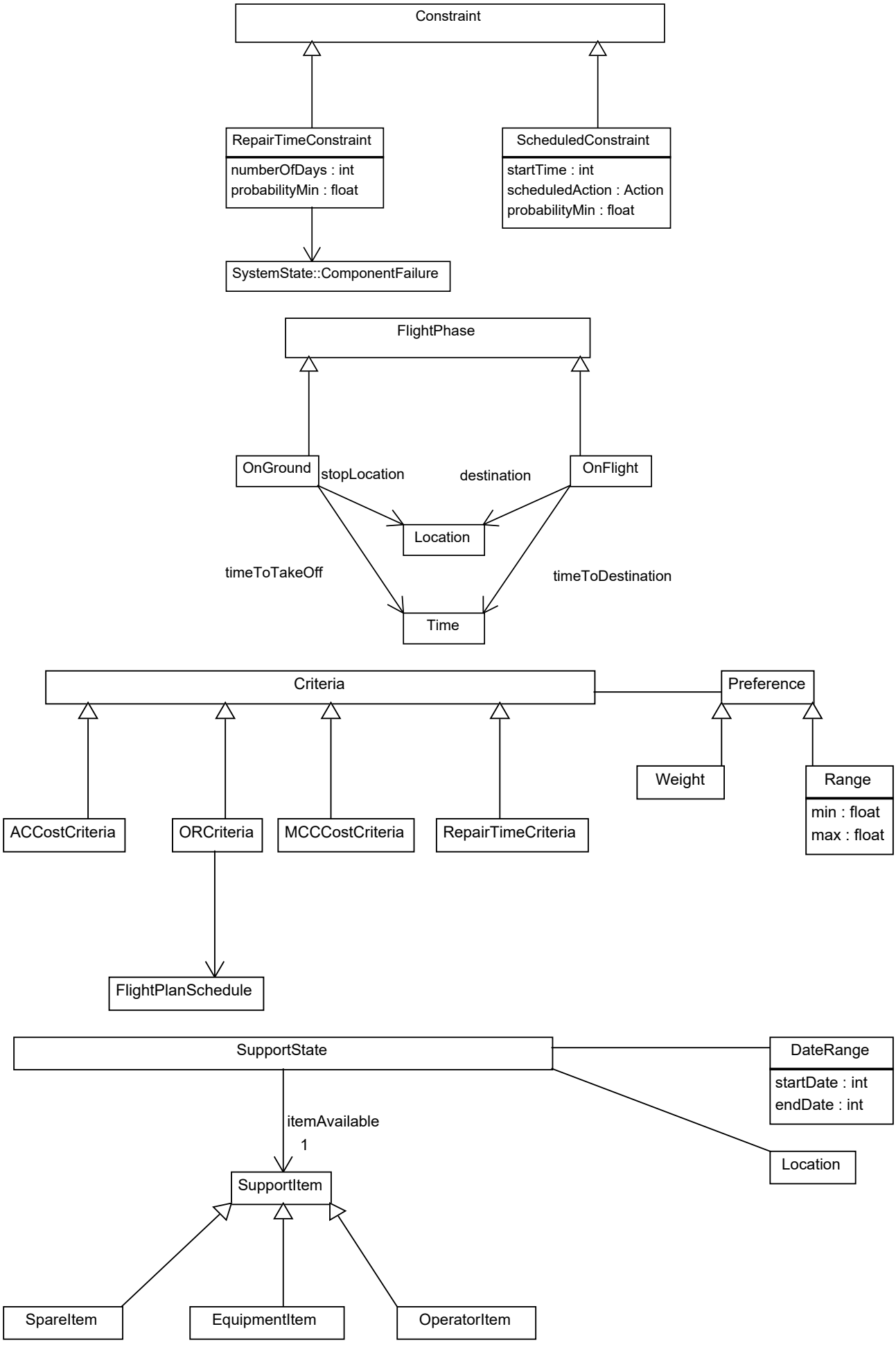
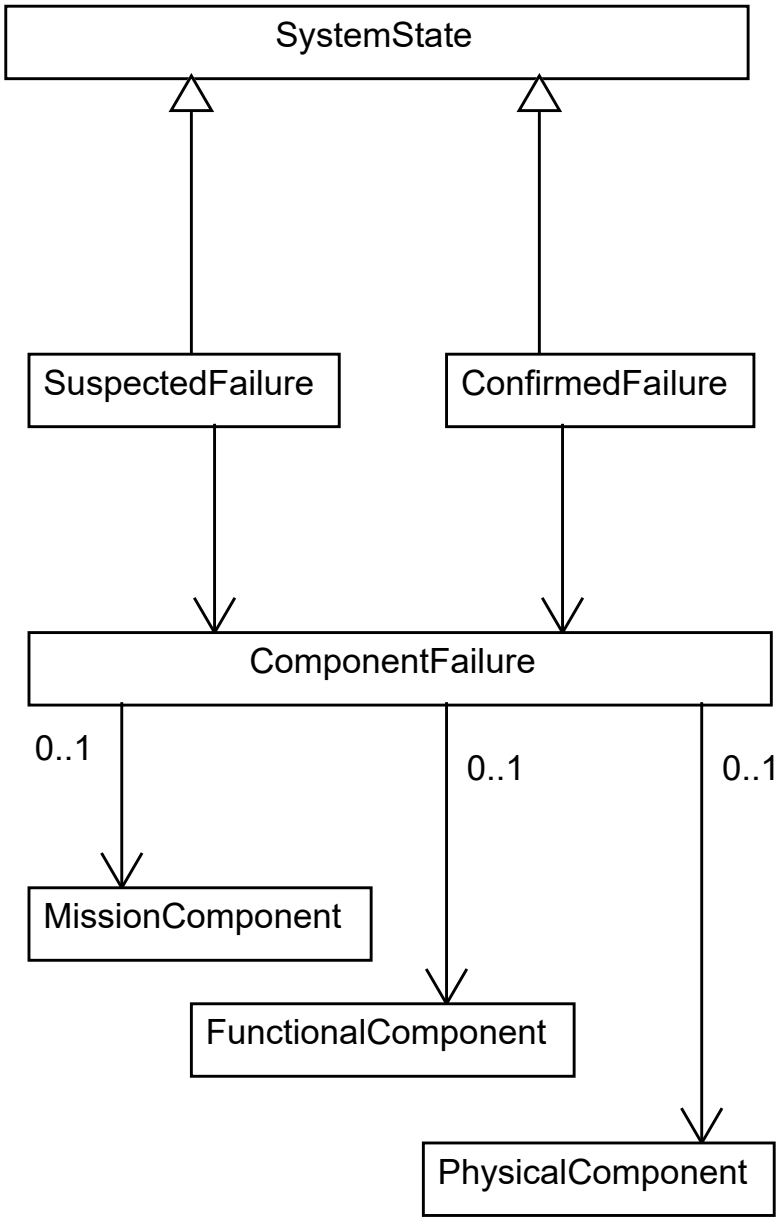


FIGURE 40 – Description UML complète du scénario Business Jet


actionLength
Action
Time
ACCost
MCCCost
RepairProcedure
actionSequence
1..*
requirement
0..*
SupportItem
RepairAction
failProbability : float
0..*
availableOutput
ReplacementAction
ReconfigurationAction
SWRepairAction
SWRebootAction
requiredFailures
1..*
repairedFailures
0..*
SystemState
ConfirmationAction
suspectedSituation
0..*
possibleResults
1..*
confirmedSituation
testProbability : float
0..*
MissionOrders
AllowDiversion
destination : Location
previousLocation : Location
TakeOffToScheduledDestination
previousLocation : Location
nextLocation : Location
SupportOrders
SendItemFromMCC
item : SupportItem
TakeItemToDestination
previousLocation : Location
destination : Location
item : SupportItem
UseItemFromAssociate
itemLocation : Location
item : SupportItem


RankedSolutions
rankedList
Plan
planTree
PlanStep
expectedState : State
correspondingAction : Action
possibleNextSteps
Explanation
recognizedDrawbacksSet
Drawbacks
UnusualCost
UnusualOperations
UnrespectedConstraints
mainOperations
Operation
OperationLocation : Location
concernedAction : Action
MostProbableTrace
probability : float
traceSequence
PlanTraceStep
expectedState : State
correspondingAction : Action


Constraint
RepairTimeConstraint
numberOfDays : int
probabilityMin : float
ScheduledConstraint
startTime : int
scheduledAction : Action
probabilityMin : float
SystemState::ComponentFailure
FlightPhase
OnGround
stopLocation
destination
OnFlight
Location
timeToTakeOff
timeToDestination
Time
Criteria
Preference
Weight
Range
min : float
max : float
ACCostCriteria
ORCriteria
MCCCostCriteria
RepairTimeCriteria
FlightPlanSchedule
SupportState
DateRange
startDate : int
endDate : int
itemAvailable
1
Location
SupportItem
SpareItem
EquipmentItem
OperatorItem


SystemState
SuspectedFailure
ConfirmedFailure
ComponentFailure
0..1
0..1
0..1
MissionComponent
FunctionalComponent
PhysicalComponent

ANNEXE C

EXEMPLE DE FICHIER PPDDL GÉNÉRÉ ALÉATOIREMENT POUR LE PROBLÈME DU BUSINESS JET

Algorithm 15: Fichier PPDDL généré pour le problème du Business Jet : Domain

```
(define (domain business-plane)
  (:requirements :typing :equality :negative-preconditions :disjunctive-preconditions
  :universal-preconditions :conditional-effects :mdp :fluents)
  (:types location system)

  (:predicates
    (at ?loc - location) (failed ?s - system) (repairable ?s - system ?loc -location)
    (in-flight ?l1 ?l2 - location) (true)
  )
  (:functions (ft ?s - system) (next ?l1 ?l2 - location))

  (:action take-off
    :parameters (?l1 ?l2 - location)
    :precondition (and (at ?l1) (> (next ?l1 ?l2) 0))
    :effect (and (not (at ?l1)) (in-flight ?l1 ?l2))
  )

  (:action fly
    :parameters (?l1 ?l2 - location)
    :precondition (and (in-flight ?l1 ?l2))
    :effect (and
      (when (not (< (next ?l1 ?l2) 2)) (decrease (next ?l1 ?l2) 1))
      (when (< (next ?l1 ?l2) 2) (and (not (in-flight ?l1 ?l2)) (at ?l2)))
      (forall (?s - system) (and
        (when (failed ?s) (increase (ft ?s) 1))
        (when (not (failed ?s)) (probabilistic 1/5 (and (failed ?s) (assign (ft ?s) 0))))
      ))
    )
  )

  (:action repair
    :parameters (?s - system ?loc - location)
    :precondition (and (failed ?s) (at ?loc) (repairable ?s ?loc))
    :effect (and (not (failed ?s)) (decrease reward 1))
  )
)
```

Algorithm 16: Exemple d'instance PPDDL générée aléatoirement pour le problème du Business Jet

```
(define (problem business-plane-3)
  (:domain business-plane)

  (:objects l1 l2 l3 l4 l5 l6 l7 l8 l9 l10 endLoc - location
    s1 s2 s3 s4 s5 s6 s7 s8 - system
  )

  (:init (at l1)
    (= (next l1 l2) 2) (= (next l2 l3) 1) (= (next l3 l4) 1)
    (= (next l4 l5) 1) (= (next l5 l6) 1) (= (next l6 l7) 1)
    (= (next l7 l8) 1) (= (next l8 l9) 1) (= (next l9 l10) 1)
    (= (next l10 endLoc) 1)
    (repairable s1 l1)  (repairable s1 l2)  (repairable s1 l3)  (repairable s1 l4)
    (repairable s1 l7) (repairable s1 l9)
    (repairable s2 l1)  (repairable s2 l2)  (repairable s2 l3)  (repairable s2 l4)
    (repairable s2 l7) (repairable s2 l9)
    (repairable s3 l1)  (repairable s3 l2)  (repairable s3 l3)  (repairable s3 l4)
    (repairable s3 l7) (repairable s3 l9)
    (repairable s4 l1)  (repairable s4 l2)  (repairable s4 l3)  (repairable s4 l4)
    (repairable s4 l7) (repairable s4 l9)
    (repairable s5 l1)  (repairable s5 l2)  (repairable s5 l3)  (repairable s5 l4)
    (repairable s5 l7) (repairable s5 l9)
    (repairable s6 l1)  (repairable s6 l2)  (repairable s6 l3)  (repairable s6 l4)
    (repairable s6 l7) (repairable s6 l9)
    (repairable s7 l1)  (repairable s7 l2)  (repairable s7 l3)  (repairable s7 l4)
    (repairable s7 l7) (repairable s7 l9)
    (repairable s8 l1)  (repairable s8 l2)  (repairable s8 l3)  (repairable s8 l4)
    (repairable s8 l7) (repairable s8 l9)
    (= (ft s1) 0) (= (ft s2) 0) (= (ft s3) 0) (= (ft s4) 0) (= (ft s5) 0)  (= (ft s6) 0)
    (= (ft s7) 0) (= (ft s8) 0)
    (true)
  )

  (:goal (or
    (at endLoc)
  ))
  (:pctl
    (true)
    (not (or (> (ft s1) 2) (> (ft s2) 2) (> (ft s3) 2) (> (ft s4) 2) (> (ft s5) 2) ))
    > 0.1
  )
  (:metric maximize (reward))
)
```

ANNEXE D

CONTEXTE SUR LA NOTION DE QUALITÉ DE SERVICE

Comme nous l'avons évoqué dans les chapitres précédents, la caractéristique qui nous intéresse le plus dans le contexte avionique est la *criticité* : lorsqu'un agent doit prendre une décision sur un système critique, il est nécessaire que son processus de décision offre des garanties fortes sur le respect de certaines contraintes, ce qui implique que ce processus doit avoir une forme particulière mettant l'accent sur la validation.

Nous avions identifié, dans la première partie, un cas de décision sûre et optimale dans la maintenance avionique, en exploitant une connaissance précise du domaine pour déterminer l'ensemble des données impliquées dans le processus de décision ; ces données étant hétéroclites, telles que la MEL ou les coûts des réparations, nous avions pu en déduire que la prise de décision devait effectivement respecter à la fois des critères d'optimalité et des contraintes de sécurité.

Bien que ce type de réflexion puisse être mené à bien sur d'autres exemples, nous pouvons considérer une autre approche : puisque la prise de décision est effectuée par un agent, il est possible de catégoriser les problèmes de décision dans le monde avionique en fonction de l'agent. Nous allons réaliser cette catégorisation dans les paragraphes suivants, au travers de l'identification de trois agents principaux :

- Le **pilote**, ou plus précisément l'**opérateur** si nous considérons un cadre plus large tel que la commande à distance.
- Le **concepteur du système**.
- Le **gestionnaire de mission** au niveau de la flotte.

Notons qu'il serait possible d'identifier de nombreux autres rôles, plus spécifiques, tels que l'opérateur de tour de contrôle, l'agent de maintenance, voire la compagnie aérienne. Néanmoins, nous avons isolé ces trois rôles puisqu'ils nous semblent recouvrir à première vue l'ensemble des problématiques de décision qu'il est possible de rencontrer, comme nous l'expliciterons dans les paragraphes suivants.

Dans cette annexe, nous étudierons ainsi un ensemble de scénarios de décisions, catégorisés selon l'agent prenant la décision, afin de présenter la démarche qui nous a amené à construire un modèle formelle de la qualité de service.

D.1 Aide à la décision destinée au pilote

Le pilote, ou l'opérateur, est évidemment l'agent principal auquel nous pensons lorsqu'il s'agit de parler de problématique de décision. Le problème de la décision, en particulier sous l'angle d'une catégorisation des tâches nécessaires au pilotage, a déjà été étudié de façon extensive par la communauté des facteurs humains ; le lecteur intéressé pourra se référer en particulier à des travaux tels que [EFJ+98], [CMF96], [Fun91], [AD92] et [Piz13].

Ces catégories ne sont cependant que d'un intérêt modéré pour notre problème, puisqu'ils offrent une vision avec un niveau de détail élevé, tandis que nous souhaitons obtenir des résultats applicables à un champ plus large : nous souhaitons déterminer dans quelle mesure les problèmes de décision que rencontre le pilote ont les mêmes caractéristiques que des problèmes de conception sûre et optimale, c'est-à-dire dans quelle mesure les décisions de pilotage doivent respecter des critères d'optimisation et des contraintes de sécurité ; nous souhaitons par ailleurs déterminer dans quelle mesure les décisions de pilotage peuvent être résolues avec des outils existants, ou peuvent être résolues en adaptant nos travaux réalisés lors des parties précédentes. Cependant, pour cela nous devons formaliser des scénarios qui concernent potentiellement plusieurs tâches de pilotage : des problèmes de décision sûre et optimale peuvent par exemple survenir lorsque le système vient de subir une défaillance et que le pilote doit agir sur plusieurs tâches pour poursuivre la mission.

Nous souhaitons donc nous baser sur ces travaux existants pour construire une vision abstraite globale, c'est-à-dire une vision qui isole des caractéristiques (continu/discret, déterministe/probabiliste, décision unique/séquentielle,...) sur le type de données et d'actions que doit traiter le pilote, de la même façon que ce que nous avons pu faire pour le problème du business jet. Nous étudierons ainsi dans les paragraphes suivants plusieurs approches nous permettant de construire une vision transverse des problèmes de décision auxquels le pilote doit faire face.

D.1.1 Pré-requis à la décision : Situation Awareness

Une approche particulièrement populaire sur la décision est celle de Endsley [End95], dont les travaux étudient le schéma cognitif suivi par le pilote lors de la prise de décision. Ces travaux sont particulièrement intéressants pour nous, puisqu'ils représentent un schéma possible pour les premières étapes d'un processus outillé assistant à la décision qui peut être mis en œuvre vis à vis du pilote.

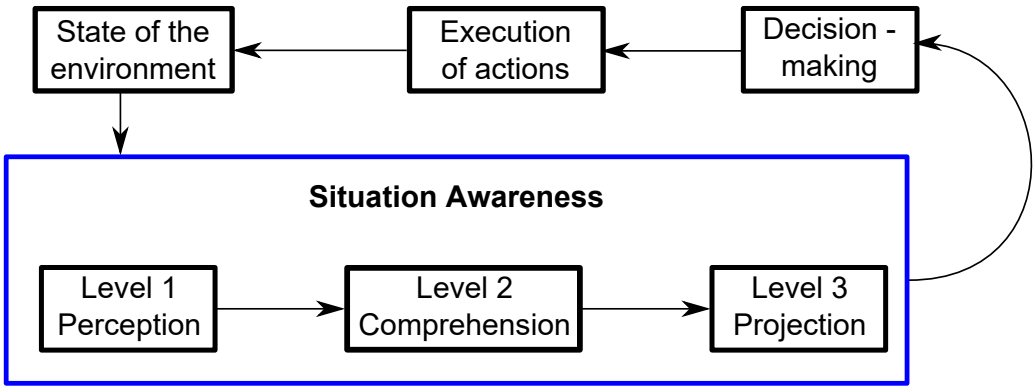


FIGURE 41 – Situation Awareness : le pilote effectue plusieurs étapes cognitives avant la prise de décision

En nous basant sur le schéma de Endsley (Figure 41 page 214), nous pouvons isoler trois étapes préliminaires à la décision qui peuvent être assistées par un processus outillé :

1. La **perception**, consistant pour le pilote à capturer les différents signaux extérieurs tels que les messages apparaissant sur le cockpit, les données de vol voire simplement ce qu'il aperçoit au travers de la vitre du cockpit.
2. La **compréhension**, où le pilote utilise les signaux perçus pour construire une représentation interne de l'état du système.

3. L'**anticipation**, où le pilote se représente les évolutions possibles du système afin d'évaluer l'impact de ses actions dans la durée.

Ces trois étapes peuvent effectivement être assistées par un processus outillé : des outils comme par exemple ceux sur lesquels s'appuie le Flight Warning utilisent la connaissance du système pour ne transmettre au pilote que les messages importants en fonction du contexte, ce qui permet de réduire sa charge lors de la perception ; un diagnostic approprié permet d'assister à la phase de compréhension, c'est-à-dire à la mise en commun des différents signaux pour construire une représentation unique de l'état du système ; enfin, dans une certaine mesure, le Flight-Manager et les outils de pronostic assistent par exemple à la phase d'anticipation, en représentant pour le pilote l'évolution temporelle de l'avion sur le plan de vol et l'évolution future des différents systèmes. La problématique de l'aide à la décision pour le pilote peut donc être divisée selon trois axes : l'aide à la perception, l'aide à la compréhension et l'aide à l'anticipation.

Notons cependant que ces trois axes ne doivent pas être traités séparément : afin d'assister le pilote dans sa phase d'anticipation, un processus outillé aura vraisemblablement besoin de modifier la manière dont la perception et la compréhension sont effectuées, par exemple au travers d'une mise en valeur de certaines informations ou d'un traitement préliminaire permettant de lever des ambiguïtés lors de la compréhension. Dans le cadre de notre étude, ces trois axes nous permettent en revanche d'établir qu'un processus outillé d'assistance aux décisions de pilotage devra nécessairement participer a chacune de ces trois étapes successivement. L'ensemble de ces étapes permet alors de construire la **Situation awareness**, c'est-à-dire une image mentale de la situation ainsi que de son déroulement dans l'avenir.

D.1.2 Problématiques de décision : redéfinir le rôle du pilote

Nous avons dans les paragraphes précédents rappelé les travaux de Endsley [End95], que nous pouvons appliquer directement pour identifier des caractéristiques sur les premières étapes d'un processus outillé d'aide à la décision pour le pilote - en mettant en avant le fait qu'il devrait suivre un chemin parallèle au chemin cognitif suivi par le pilote. Dans les paragraphes suivants, nous nous baserons sur une analyse naïve pour soulever quelques considérations notables sur la suite du processus outillé.

De façon concrète, considérons à présent la formulation suivante des considérations de pilotage :

Exemple de considérations de pilotage

1. Le pilote, ou l'opérateur, a pour objectif principal de **terminer la mission** fixée,
2. en respectant les contraintes imposées par la **réglementation des espaces aériens** et les réglementations environnementales,
3. ainsi que les **contraintes de temps** imposées par la compagnie aérienne,
4. en maximisant le niveau de **disponibilité** de l'avion suite à la fin de mission,
5. en utilisant un **minimum de carburant**,
6. dans la mesure du possible en un minimum de temps (sous respect des contraintes précédentes),
7. dans la mesure du possible avec le plus de confort pour les passagers,
8. ainsi que dans la mesure du possible avec le plus de considération pour l'environnement traversé.

Cette liste est volontairement établie de façon naïve : elle résulte d'une lecture sommaire de l'état de l'art existant (en particulier [EFJ+98]), que nous avons évoqué précédemment, ainsi que d'une connaissance du milieu - sans être une connaissance d'expert. Tout comme les autres listes ou schémas de considérations existant dans l'état de l'art, elle a aussi le défaut principal de décrire un état de fait,

et non un objectif de développement : nous disons subrepticement au travers de cette liste que le rôle du pilote est d'effectuer la gestion du carburant, alors que ce rôle est de moins en moins réalisé par le pilote, voire de moins en moins réalisable par le pilote. Face à la complexité existante, le pilote (ou plus généralement l'opérateur) s'est retrouvé privé dans les faits de ses rôles traditionnels de navigation, de gestion de mission et de gestion des systèmes, au profit de systèmes automatisés qui peuvent gérer cette complexité.

Nous pouvons faire deux remarques face à ce constat :

1. Le rôle du pilote est sensiblement différent en fonction de la complexité du système.
2. Bien qu'il ne semble pas adapté aux nouvelles conditions de pilotage, le pilote est toujours considéré comme un acteur indispensable (et responsable) par l'ensemble de l'industrie aérienne.

En effet, il semble évident qu'un avion de ligne et un business jet imposent des charges différentes au pilote, puisque les équipements disponibles n'ont pas la même complexité ; ceci est particulièrement vrai puisque la durée de vie des systèmes avioniques et des avions est extraordinairement longue, en comparaison avec la vitesse d'évolution des technologies : l'Airbus A320 voit par exemple cohabiter pour l'instant 4 versions (ainsi que plusieurs versions en développement [www]) dont la première date de 1987. Par conséquent, toutes les questions d'automatisation, de rôle du pilote ainsi que d'assistance à la prise de décision doivent être considérées relativement à un avion donné, avec ses équipements ou ses possibilités d'ajout d'équipements : si l'on souhaitait automatiser entièrement un A320, il faudrait que le système d'automatisation puisse par exemple disposer de toutes les informations nécessaires pour la prise de décision. Ceci semble de toute évidence difficilement réalisable, en particulier puisque les systèmes de guidage existants ne sont pas faits pour être accessibles depuis un équipement extérieur.

Afin de restreindre notre champ d'étude, nous faisons donc l'hypothèse de travail que le système d'aide à la décision serait installé dans un nouvel avion, ou tout du moins un avion pour lequel nous n'avons pas de contraintes pour installer de nouveaux équipements. La question des avions existants est jugée trop spécifique et trop restrictive pour permettre une analyse pertinente de notre processus outillé, bien qu'elle soit évidemment tout aussi intéressante, en particulier sur le plan économique.

La seconde remarque pose en réalité la question suivante : pourquoi les avions ne sont-ils pas encore entièrement autonomes ? Il est de notoriété publique que beaucoup de fonctionnalités dans les avions modernes sont automatisées, telles que le vol, le décollage, l'atterrissage ainsi que la gestion de défaillances mineures. L'actualité a par ailleurs participé à nourrir notre réflexion à ce sujet au cours de notre étude : l'année 2015 a vu se jouer le drame du crash aérien d'un airbus A320 dans les Alpes, les premiers rapports fournissant comme explication des troubles psychologiques de la part du pilote [BEA] ; mais cette année a aussi vu la popularisation dans les médias des voitures sans pilotes, des drones de loisir, ainsi que la publication de plusieurs rapports annonçant une pénurie à venir de pilotes dans les prochaines années [oT], particulièrement en raison du contexte économique des compagnies aériennes ainsi que de plusieurs décisions rendant l'industrie peu attractive pour les jeunes employés.

La principale raison énoncée par les experts est qu'il s'agit d'un problème complexe : les statistiques évoquent près de 40% de crashs aériens pour les drones militaires américains [Pos] ; seulement un faible pourcentage de tous les atterrissages sont réalisés de façon automatique, principalement parce que la configuration du système d'atterrissage automatique est tout aussi complexe que l'atterrissage manuel - hormis dans les cas où les conditions météorologiques réduisent grandement la visibilité ; enfin, des exemples récents apportent des arguments à la fois en faveur et en défaveur de l'automatisation, par exemple le cas du vol US Airways 1549 [Boa] que les pilotes ont pu faire atterrir sur la rivière Hudson, ce qui n'aurait pas pu être possible avec l'auto-pilote seul ; ou à l'inverse le cas du vol Air France 447 où l'auto-pilote a rendu la main aux pilotes, dont les mauvaises réactions ont contribué à la perte de tous les passagers. Loin d'être des arguments contre l'autonomie complète, ces justifications sont autant d'arguments pour le fait que l'autonomie n'est pas encore un problème résolu : ceci prouve que les systèmes actuels ne suffisent pas à faire une autonomie complète, et qu'il n'existe pas de solution triviale pour franchir le pas.

Ceci est appuyé notamment par un second argument avancé par les experts, qui est un argument économique : l'application de technologies de contrôle à distance ou d'autonomie complète représente un coût d'investissement important pour l'industrie entière, depuis les fabricants de composants aux

compagnies aériennes. Ce coût recouvre en particulier des investissements d'infrastructures au niveau des aéroports et des tours de contrôle de trafic, un coût potentiel lié au report de la responsabilité sur le concepteur de l'appareil en cas d'accident, ainsi qu'un investissement non négligeable de communication auprès d'un public réticent à monter dans un avion sans pilote ; concernant la voiture autonome - qui semble réalisable à un horizon plus proche que l'avion autonome - les sondages de la presse évoquent une proportion de 35% de personnes qui accepteraient de monter dans une voiture autonome, 65% déclarant qu'elles se sentiraient mal à l'aise sans conducteur [Odo]. Ceci est à ajouter au fait que l'industrie avionique met en œuvre des processus longs, tels que les processus de certification : si un fabricant débutait aujourd'hui un mouvement général vers l'avion autonome, le retour d'expérience sur les anciens programmes nous montre qu'il s'écoulerait entre 10 et 15 ans au minimum avant de voir le premier avion autonome commercial voler.

Enfin, le troisième argument réside dans les plus-values que seul un pilote peut apporter. On pense notamment à la gestion de problèmes imprévus, tels que des problèmes internes à l'avion (un passager faisant un malaise), des problèmes externes et imprévisibles (des passagers trop longs à embarquer), des erreurs dans le déroulement de procédures prévues (l'équipage oubliant de fermer une porte), ou encore l'évaluation d'une situation pour laquelle il n'est pas trivial de créer une règle prédéfinie (le contournement d'un cumulus, à quelle distance, pendant combien de temps, ...). Ce qu'on recouvre par cet argument est la gestion des urgences qui *ne sont pas dans le livre*, faisant appel à la capacité d'analyse et d'expérimentation du pilote. C'est une thématique qui a été à nouveau mise en avant récemment avec les questions de sécurité face aux piratages informatiques des systèmes : lorsque le système de pilotage autonome est lui-même défaillant ou corrompu, la présence d'un pilote à bord est la seule solution existante permettant d'assurer un minimum de contrôle.

Ce que nous devons conclure de cette seconde remarque n'est donc pas un jugement de valeur sur l'autonomie complète de l'avion : il est difficile, sinon futile, de déterminer si l'autonomie complète est un bien ou un mal pour l'industrie avionique ; il est en revanche clair que dans un horizon immédiat le pilote gardera un rôle important dans la conduite de l'avion, et que l'automatisation des systèmes doit prendre en compte cet état de fait.

Dans les paragraphes suivants, nous nous appuierons sur ces considérations pour détailler la suite des caractéristiques d'un processus outillé d'aide à la décision pour le pilote, en particulier en étudiant quelles décisions ou informations bénéficieraient d'une automatisation ou d'une prise en charge par le pilote.

D.1.3 Problématiques de décision : répartition des décisions entre systèmes autonomes et pilote

Reprenons la liste naïve des considérations de pilotages que nous avons établie lors des paragraphes précédents. Parmi cette liste, nous souhaitons identifier puis classer les considérations en deux catégories : celles qui gagnent à être traitées de façon entièrement automatisée et celles qui bénéficient au contraire d'un traitement par le pilote.

Nous avons déjà évoqué précédemment le fait que la gestion du carburant est fastidieuse, sinon irréalisable par le pilote dans les avions modernes. Ceci est par ailleurs le cas de la gestion du temps de trajet, ainsi que des aspects écologiques et du confort des passagers : il s'agit de problèmes pour lesquels trop d'informations sont mises en jeu, et qui ne nécessitent pas les capacités d'analyse du pilote lorsque l'avion fonctionne de façon nominale ; par ailleurs, lorsque l'avion est en fonctionnement dégradé, ces considérations passent au second plan - il serait presque préjudiciable que le pilote doive gérer les économies de carburant alors qu'il a d'autres tâches de pilotages plus pressantes dues à l'état dégradé de l'avion. Ces considérations bénéficient donc clairement d'une automatisation complète, soit sous la forme d'une synthèse des informations pour informer le pilote lors de son processus cognitif (avertir des différentes options de plan de vol en fonction de l'économie du carburant), soit en prenant de façon autonome un ensemble de décisions participant à ces considérations, telles que des optimisations dans les algorithmes de guidage ou des décisions automatiques de reconfiguration pour l'économie d'énergie.

Les contraintes de réglementation (espace aérien, environnement, temps) sont aussi fastidieuses à gérer pour un pilote, puisqu'elles mettent en jeu de nombreux paramètres, mais sont dans une certaine mesure autant difficiles à gérer pour un système autonome puisque l'infrastructure existante

ne permet pas de communiquer ces contraintes directement à l'avion. C'est par exemple le cas du chemin de roulage, qui est communiqué par la tour de contrôle au pilote, lequel doit alors configurer les équipements de bords en fonction de ces consignes. Il s'agit de considérations qui auraient un bénéfice clair à être automatisées, en particulier puisque l'optimisation des chemins de roulage a déjà été traité par plusieurs études [VR03] [SS04] [Mar06], mais qui ne le seront vraisemblablement pas dans un horizon immédiat pour des raisons d'infrastructure. De plus, ces contraintes doivent être assurées même en cas de défaillance du système, ce qui implique que même avec une automatisation complète de leur gestion, le pilote devra être suffisamment impliqué dans le processus de décision pour pouvoir reprendre la main rapidement en cas de défaillance. La solution est alors à court terme de proposer des systèmes d'aide à la décision, plutôt que des systèmes autonomes, qui proposent aux pilotes des scénarios possibles tout en leur laissant la décision finale.

Enfin, les deux dernières considérations peuvent être formulées de façon générale comme étant de terminer la mission avec un avion dans le meilleur état possible. S'il est évident que ceci repose en grande partie sur des systèmes automatisés, c'est ici que le rôle du pilote comme redondance supplémentaire est aujourd'hui indispensable. En effet, lors d'un fonctionnement nominal [1] de l'avion, ces objectifs sont remplis par les systèmes de pilotage automatique déjà présents à bord des avions commerciaux ; en revanche, dès lors que des défaillances non triviales surviennent, le pilote peut être amené à utiliser sa capacité d'analyse et d'expérimentation pour pallier la défaillance du système. Nous avons ainsi isolé une catégorie de décision où il est indéniable que le pilote a un rôle prédominant à l'heure actuelle : il s'agit des décisions effectuées de façon *réactive suite à une défaillance* afin de *remettre l'avion en état de poursuivre la mission.*

D.1.4 Conclusion sur la catégorisation et abstraction du problème

Dans les paragraphes précédents, nous avons pris comme point de départ une analyse de l'état de fait actuel des considérations de pilote. Nous avons construit une image (basée sur la perception nécessairement subjective des projets publics existants) de l'évolution des équipements d'automatisation à court et moyen termes, nous permettant d'évaluer - dans le contexte de la création d'un nouvel avion et dans le contexte industriel actuel - quelles parties des considérations de pilotages étaient concernées par des systèmes d'automatisation et quelles parties relevaient plutôt d'une aide à la décision au pilotage. Nous avons ainsi pu identifier plusieurs catégories et nous avons évoqué certaines des caractéristiques de ces catégories. Notons que, bien que nous nous soyons basés sur une analyse "naïve" des considérations de pilotage, l'exercice aurait pu être mené sur une analyse plus approfondie, en partant des travaux existant dans la littérature [EFJ+98].

Nous pouvons tirer plusieurs leçons de cette réflexion, comme par exemple le fait que plus une décision est critique et plus le pilote semble devoir garder un rôle prépondérant dans la décision, ou à l'inverse que plus une décision implique un grand nombre d'informations et plus elle bénéficie d'une automatisation partielle ou complète. Néanmoins, les conclusions de ce type étant obtenues à partir d'une réflexion générale, basée sur une évaluation naïve des considérations du pilote, elles demeurent des conclusions obtenues par retour d'expérience, et non prouvées par une analyse exhaustive et un cheminement logique précis. Ce que montrent finalement les paragraphes précédents est qu'une catégorisation des décisions, même précise, ne permet de tirer que des conclusions mineures sur la nature du problème de décision.

A l'inverse, une approche centrée sur la notion de **qualité de service** apparaît ici comme particulièrement pertinente ; comme nous l'avons évoqué dans les chapitres précédents, ces exemples mettent en avant la considération suivante :

> Toutes les décisions prises par le pilote visent à améliorer un ensemble de paramètres, qui sont caractérisés par le fait qu'ils sont mesurables (le pilote peut se rendre compte que ses actions améliorent ou déprécient chaque paramètre), objectifs (deux pilotes différents verront tous les deux de façon manifeste - en dehors des problèmes de perception - lorsqu'un paramètre augmente), munis d'un ordre total (le paramètre peut être clairement identifié

1. normal, dans le jargon avionique

comme étant amélioré ou déprécié) et munis d'une borne supérieure dans le sens de l'amélioration (il existe une valeur *idéale* de chaque paramètre). Notons que ceci ne présage en rien de la manière dont le pilote prendra la décision : il peut privilégier l'amélioration de certains paramètres par rapport à d'autres selon une logique qui lui est propre.

Nous rejoignons alors le type de réflexion que nous avons mené précédemment sur la définition informelle de la notion de qualité de service.

Au travers d'une réflexion sur le rôle du pilote dans le cadre de la décision, nous avons donc identifié le fait que la gestion de la qualité de service semblait être un élément essentiel de tout processus d'aide à la décision pour le pilote. Dans les parties suivantes, nous verrons que cette notion de gestion de la qualité de service est dans une certaine mesure une caractéristique de la plupart des problématiques d'aide à la décision dans le domaine avionique.

D.2 Aide à la décision lors de la conception

Dans le contexte avionique, il semble évident que la conception du système est lui-même un processus de conception sûre et optimale : il s'agit d'un processus, généralement outillé, visant à concevoir un système respectant des critères d'optimalité et des contraintes de validité. Les critères d'optimalité sont définis en termes de performance du système, tandis que les contraintes de validité sont définies comme le respect d'un certain nombre de règles imposées à la fois par des documents de spécification et par des autorités de spécification.

Notons qu'il est d'usage de faire la différence entre deux types d'évaluation du système [900] :

Définition 47 (*Vérification et Validation*)

*On parle de **Vérification** pour désigner la confirmation au travers de preuves objectives que des exigences spécifiées ont été remplies. Ceci permet donc d'établir que le produit ou logiciel a été construit correctement, vis-à-vis d'un cahier des charges.*

*On parle de **Validation** pour désigner la confirmation au travers de preuves objectives que les exigences d'usages ont été remplies. Ceci permet donc d'établir que l'application rend à un utilisateur les services exigés.*

Un processus d'aide à la décision lors de la conception est donc fortement lié au processus de Vérification et Validation du système. Ce processus repose traditionnellement sur un ensemble de processus manuels ou automatiques réalisant des **tests dynamiques**, se décomposant par exemple en plusieurs catégories telles que les **tests unitaires** (évaluant une unité élémentaire), les **tests d'intégration** (évaluant la combinaison de plusieurs unités élémentaires), les **tests systèmes** (évaluant le système complet) ou encore les **tests d'acceptance** (évaluant si le système sera accepté par l'utilisateur).

Cependant, la vérification et validation (V&V) d'un système ne présentent pas en elles-mêmes de composante de décision : les processus de V&V sont des outils de vérification d'un système existant, et non de création de ce même système.

On parle **d'ingénierie système** pour désigner les approches de conception des systèmes complexes, dont les champs principaux sont la gestion des exigences, l'architecture fonctionnelle et logique, et l'intégration/vérification/validation/qualification. Il s'agit d'un domaine vaste, dont les principales considérations ont été détaillées dans différentes études auxquelles un lecteur intéressé pourra se référer [Wym93] [FLV06].

L'une des remarques importante à retenir de l'état de l'art existant est que la complexité croissante des systèmes cause la généralisation de l'utilisation d'outils de modélisation pour les processus d'ingénierie système : de nombreuses études ont montré que l'utilisation de modèles permettait de gérer efficacement la complexité d'un système, en permettant d'estimer l'efficacité d'un système, sa performance ainsi que ses caractéristiques techniques. Notons que dans la littérature, le terme *Modèle* désigne quelque chose de plus riche qu'un simple dessin ou diagramme, puisqu'un modèle permet par exemple d'effectuer des simulations du système. De la même façon, il est important de souligner qu'un système est rarement représenté par un modèle unique, mais plus généralement représenté par plusieurs modèles ; chacun de ces modèles présente alors un ensemble de relations entre des entrées et des sorties du système, relations qui peuvent être aussi simples que l'ajout de quantités ou alors aussi complexes que des équations différentielles.

À partir de cette réflexion, nous pouvons donc en conclure que, dans le cadre de la conception des systèmes, parler de processus de conception sûre et optimale revient simplement à parler de processus d'ingénierie système basé sur des modèles et générant automatiquement des preuves de Vérification et Validation. Dans la mesure où ce processus est outillé, l'aspect "optimal" vient alors du fait que l'outil permet automatiquement d'optimiser certains paramètres du système, tout en garantissant évidemment que les contraintes de vérification et validation sont respectées.

Définissons plus en avant les paramètres à optimiser : il s'agit de paramètres permettant de définir que le système remplit bien l'usage qui est attendu de lui de la part d'un utilisateur. Ces paramètres sont donc, dans tous les sens de cette caractérisation, des qualités de service. De la même façon qu'une décision de pilotage est le résultat d'un critère reposant sur les qualités de service fournies par l'avion,

il est possible de définir un *bon* et un *mauvais* système sous la forme d'un critère reposant sur les qualités de service. Ceci est naturel lorsqu'on garde en mémoire la définition, très générale, des qualités de service : nous venons simplement d'établir que la qualité d'un système est évaluée en termes de la capacité de toutes ses sorties à être conformes à un certain paramètre de qualité.

Ceci est cependant tout aussi vrai des contraintes de validation, qui peuvent aussi être exprimées en termes de qualité de service : vérifier le bon fonctionnement d'un système peut uniquement être effectué au travers de tests sur les sorties de ce système (en connaissant les entrées) qui doivent respecter certaines contraintes ; ainsi, le fait de vérifier que le système n'arrive jamais dans une situation redoutée - ou n'arrive dans une situation redoutée qu'avec une très faible probabilité - peut être exprimé sous la forme d'une vérification que la qualité de service demeure toujours au dessus d'un niveau minimal.

Nous pouvons à nouveau voir que la notion de qualité de service est utile, sinon centrale, pour exprimer les critères et contraintes de décision. Il est important de noter que ceci n'implique pas que l'expression des qualités de service fournies par un système soit le seul moyen de concevoir un système d'aide à la décision - ni qu'il soit le meilleur. Nous pouvons simplement conclure que l'expression des qualités de service semble être un moyen approprié et réalisable pour la définition de critères et contraintes de décision.

Enfin, outre cet argument pour l'utilité d'une expression des qualités de service d'un système, la seconde conclusion que nous pouvons tirer de cette réflexion est qu'il existe une très forte proximité entre les critères d'optimalité et les contraintes de validité. Plus précisément, nous avons évoqué précédemment le fait que parler de processus de conception sûre et optimale dans le cadre de l'ingénierie système désignait simplement un processus d'ingénierie système basé sur des modèles, générant automatiquement des preuves de validation et vérification ; si le modèle est exprimé sous la forme d'un ensemble d'informations sur la qualité de service, alors nous pouvons en déduire que ce processus a pour objectif de modéliser la propagation de la qualité de service au sein du système, permettant à chaque étape de la conception de confronter ces informations sur la qualité de service avec des critères d'optimalité et des contraintes de validité.

D.3 Aide à la décision lors de la planification de mission

Il est possible d'étudier les problèmes de décision qui sont posés par les autres acteurs que le pilote et le concepteur de l'avion ; ceci recouvre par exemple la compagnie aérienne, responsable de la gestion d'une flotte entière, les tours de contrôle ou encore les opérateurs de maintenance. Nous avons déjà étudié dans une partie précédente les considérations propres à la maintenance, et montré que ces considérations pouvaient être vues sous l'angle d'un problème d'aide à la décision appuyé par un processus outillé de conception sûre et optimale ; de la même manière, les compagnies aériennes sont confrontées à des décisions similaires, puisque la gestion de la flotte revient à l'agrégation de plusieurs problèmes individuels de gestion d'avions.

Néanmoins, la gestion de la flotte doit prendre en compte des données qui sont plus complexes à modéliser que les données de maintenance : par exemple, une compagnie aérienne doit être capable de réagir à des retards causés par des passagers ; ou encore d'évaluer l'impact de certains paramètres météorologiques sur la suite du vol. Ainsi, bien qu'il serait possible d'étudier plus en avant les scénarios d'aide à la décision dans le cadre de la gestion de flotte, nous ne réaliserons pas cette étude puisque nous pouvons d'ores et déjà établir que les données d'entrée présentent le même type d'incertitude que les données d'entrées qui étaient problématiques pour le business jet.

Concernant les problèmes d'aide à la décision dans le contexte du trafic aérien, il est important de noter que de nombreuses tours de contrôle sont déjà équipées d'installations permettant l'aide à la décision, voire l'automatisation de certaines décisions telles que le routage des avions au sol. Néanmoins, ces équipements sont loin d'être parfaits : les systèmes de contrôle de trafic aérien gèrent plus de 50 000 vols par jour, nombre augmentera vraisemblablement dans les prochaines années, ce qui implique que toute défaillance de ces systèmes informatiques pose un problème majeur et des retards conséquents. Cependant, cette criticité des équipements est loin d'être la considération principale lors de la décision : lorsqu'un contrôleur aérien prend une décision concernant le routage d'un avion, il n'a vraisemblablement pas besoin de prendre en compte la panne éventuelle de son système d'assistance, ou du système de suivi GPS ou de communication radio. Plutôt que de se concentrer sur les aspects de criticité (i.e. de contraintes de chemin devant impérativement être respectées), les études existantes s'intéressent le plus souvent à des problèmes d'optimisation sous contraintes [BP98], à des problèmes de détection et de gestion de conflit [PFB02], à des problèmes de planification ou de contrôle multi-agents [TA07], ou encore à une vision du problème sous la forme de la gestion de flux.

Plus précisément, le problème du contrôle aérien peut être envisagé sous l'angle de la gestion de flux, puisqu'un agent gérant le trafic aérien doit arriver à anticiper et gérer le volume d'avions entrants et sortants en fonction de la capacité de l'aéroport et des voies d'approches, tout en assurant une certaine robustesse vis-à-vis d'évènements imprévus tels que les retards causés par les conditions météorologiques [BP00] ou par d'autres conditions extérieures. L'approche mise en place actuellement est principalement locale, c'est-à-dire que chaque zone aérienne considère les flux entrants et sortants, prenant des décisions sur son problème local avec peu de concertation avec les autres zones aériennes. Ceci est mitigé par des processus récents tels que la réservation de créneaux [CL00], dans lequel un avion est maintenu au sol avant même le départ lorsqu'on sait que l'aéroport d'arrivée ne pourra pas l'accueillir, et qu'il devrait sinon rester en attente au dessus de l'aéroport. De façon locale, les outils d'aide à la décision consistent en un ensemble d'équipements permettant de planifier les arrivées et départs tout en maintenant le niveau d'utilisation des pistes à un optimal, ainsi qu'en un ensemble d'équipements permettant de détecter des conflits à venir, comme par exemple une collision potentielle due à un dépassement de zone.

Cette étude des cas de décision dans le cadre de la planification de mission ne nous permet donc pas, en première approximation, d'alimenter notre réflexion sur une généralisation de notre cas d'étude sous l'angle de la qualité de service : bien que ces problèmes de décision puissent être traités avec des technologies d'intelligence artificielle similaires à celles que nous avons étudiées dans les chapitres précédents, elles ne semblent pas présenter de contraintes de sécurité, qui imposent une forme si particulière aux modèles et algorithmes de résolution que nous devons utiliser dans le cadre des processus de conception sûre et optimale.

Au bilan, de la même façon que nous avions pu identifier un scénario d'aide à la décision pour la

maintenance avionique en nous basant sur des connaissances générales du domaine, nous avons pu établir au cours des paragraphes précédents que plusieurs problématiques de décision, à la fois dans un contexte de pilotage et de conception de systèmes, pouvaient être abordées comme un problème unique de gestion des qualités de service sortant d'un système. Cette démarche constitue l'argument principal justifiant le choix d'un modèle de généralisation centré sur la notion de qualité de service.

ANNEXE E

IMPLICATIONS INFORMELLES DE LA DÉFINITION DE LA QUALITÉ DE SERVICE

On rappel la définition informelle de la qualité de service suivante :

Définition 48 (*Service et Qualité de Service (rappel)*)

Un ***service*** *est une fonctionnalité mise à disposition par un système dans l'objectif d'effectuer une tâche particulière.*

Une ***qualité de service*** *est un paramètre associé à un service, représentant la performance de ce service. Ce paramètre est muni d'un ordre total et possède une valeur maximale.*

Si l'on souhaite appliquer le même type de point de vue à la notion de décision de la part du pilote, alors il devient évident que la prise de décision se base sur la perception (potentiellement partielle) d'un ensemble de qualités de service pour toutes les options possibles, suivie par le choix de la meilleure option en fonction de ces qualités de service :

Définition 49 (*Décision par la qualité de service*)

Soit un problème de décision caractérisé par un choix à effectuer entre plusieurs options $(o_1, \ldots, o_n)$. *Soit* $(s_1, \ldots, s_m)$ *un ensemble de services et* $\forall i \leq n, \forall j \leq m, q_j^{(i)}$ *la qualité de service du service* s_j *telle qu'elle serait dans l'option* o_i. *On appelle* ***décision*** *la sélection d'une option* o_i *maximisant un certain critère* $C(o_i)$.

Alors, le critère de sélection C *ne dépend que des qualités de service associées à l'option* o_i. *Il peut s'écrire sous la forme* $C(q_1^{(i)}, \ldots, q_m^{(i)})$. *Ceci signifie que la décision ne dépend d'aucun autre paramètre que des qualités de service de chacune des options :* toute décision vise à améliorer un critère de qualités de service.

Détails La formulation mathématique, bien que potentiellement incongrue dans ces paragraphes sémantiques sinon psychologiques, est utilisée ici pour clarifier la définition de décision : nous considérons comme axiome le fait qu'une décision est un choix entre plusieurs options possibles, chacune associée à des paramètres différents de qualité de service. Bien que nous ne détaillons pas ce que contiennent ces options, il est facile d'imaginer qu'elles représentent des futurs possibles, avec toutes les variables que l'on souhaite définir telles que les coordonnées de l'avion, la température, le fonctionnement de certains systèmes ou encore l'humeur de l'équipage. Dans les faits, si chacune de ces variables peut être exprimée sous la forme d'une qualité de service, alors l'hypothèse est évidente ; ceci est trivial dès lors qu'on considère un horizon temporel fini et un nombre d'options fini (ce qui est le cas ici) : il est possible de discrétiser l'ensemble des variables ayant pour domaine $\bar{\mathbb{R}}$ (les nombres réels augmentés de $+$ et - l'infini) sous la forme d'intervalles ayant du sens pour la décision, et le nombre de ces intervalles est fini, ordonné totalement et muni d'une valeur maximale. Cette démonstration prouve bien *la lettre* de ce théorème.

Prouver *l'esprit* du théorème est plus complexe, et moins mathématique : cette hypothèse exprime en effet qu'il existe un nombre réduit de paramètres (les qualités de service) tels que le pilote puisse prendre une décision tout aussi bonne en connaissant uniquement ces paramètres que s'il était omniscient - c'est-à-dire que ces paramètres suffisent à représenter exhaustivement le monde entier, dans le contexte de la décision. De plus, ces paramètres représentent (toujours dans l'esprit du théorème) une notion d'amélioration et de dégradation : une option est la meilleure lorsqu'elle améliore un certain nombre de choses de façon mesurable. Ceci n'est pas un résultat trivial, puisque par exemple ce théorème affirme que le pilote n'a pas besoin de savoir quels systèmes fonctionnent ou non dans l'avion pour prendre une décision, mais seulement si les services rendus par l'ensemble des systèmes ont une qualité suffisante. L'argument que l'on peut opposer à cette affirmation est donc qu'il existe des informations qui ne peuvent être exprimées sous la forme de qualités de service. Pour revenir à l'expression mathématique, l'argument signifie que certaines des entrées $q_j^{(i)}$ ne peuvent pas être obtenues en pratique, ou alors que la fonction de traduction de certaines variables sous la forme de qualités de service perd trop d'informations, ce qui ne permet pas d'effectuer une décision correcte.

Si on se pose la question d'une information qui n'est pas exprimable sous la forme d'une qualité de service, étant donné l'esprit de notre définition il s'agirait d'une information qui n'a pas de notion d'amélioration ou de dégradation ; ceci est par exemple le cas dans certains contextes de décision tels que la diplomatie : une décision en politique ou diplomatie peut tout à fait être basée sur des informations qui ne sont pas associées à un jugement de valeurs, voire qui ne sont pas rationnelles. Ceci nous montre une caractéristique essentielle attendue de la part du pilote : il doit prendre des décisions de façon rationnelle, c'est-à-dire fondée sur la raison, autrement dit fondée sur un "ensemble de principes, de manières de penser permettant de bien agir et juger" [Dictionnaire Larousse]. Nous pensons que cette caractéristique est manifeste ; bien que ceci ne prouve pas la justesse de cette hypothèse (dans l'esprit), puisque nous avons seulement montré que l'un des contre-arguments possible pouvait être réfuté, nous estimerons qu'elle est néanmoins suffisamment juste pour servir de base à nos réflexion futures. Notons que l'hypothèse de rationalité est à la base de plusieurs paradigmes, tels que l'hypothèse de rationalité économique qui établit que tout consommateur maximise son utilité personnelle.

L'implication principale de cette hypothèse est que la prise de décision est en réalité le résultat d'une interaction au travers de requêtes et de réponses sur la qualité de service. Ceci implique que le processus de construction de la situation awareness que nous avons détaillé précédemment (perception, compréhension, anticipation) n'est pas un processus visant à permettre au pilote de construire une représentation mentale de l'état du système et de son évolution, mais plutôt de construire une représentation mentale des qualités de service et de leurs évolutions.

Ceci implique qu'un outil d'aide à la décision pour le pilote devrait être basé sur la communication des modifications des qualités de service, et non sur l'état des composants : l'information importante pour le pilote est de savoir quelles sont les qualités de service actuelles, comment elles ont évolué ainsi que comment elles évolueront après une décision ou après une défaillance. Si le pilote disposait de ces informations avec une précision parfaite, alors la décision serait triviale, voire automatisable parfaitement. Nous pouvons, de façon audacieuse, proposer la théorie qu'une partie non négligeable des problèmes d'établissement de la situation awareness est causée par le fait que les systèmes communiquent au pilote des données autres que les qualités de service, ce qui implique qu'il doit reconstituer après-coup ces qualités de service à partir de données potentiellement parcellaires. Cette théorie a pour conséquence que le rôle du pilote est effectivement primordial dès lors où il n'est pas possible de disposer des qualités de service précises : en se reposant sur les capacités d'analyse du pilote, il est possible de reconstituer les valeurs des qualités de service manquantes à la prise de décision.

ANNEXE F

SYNTAXE RÉSUMÉE DU LANGAGE ALTARICA

Le langage Altarica repose autour de trois objets de base : **domaines, valeurs constantes** et **nœuds** (nodes).

Les domaines sont des ensembles, pouvant être infinis voire abstraits, mais devant être dénombrables. Les constantes désignent des valeurs particulières des domaines. Les nœuds vont représenter des composants, des sous-systèmes réels ou abstraits. Un nœud a plusieurs paramètres :

Variables de flux : Cet ensemble de variable représente les entrées et sorties du système. Elles permettent de connecter plusieurs composants (nœuds), comme des câbles pour des systèmes électriques. Certains dérivés du langage Altarica font explicitement la différence entre les variables d'entrée (in) et celles de sortie (out). La valeur d'une variable de flux peut dépendre à la fois de la valeur de la source où elle est connectée et de certaines variables d'état, propres au nœud dans lequel ces entrées/sorties sont définies.

Variables d'état : Cet ensemble de variables défini un automate interne. Une variable d'état représente un état interne du système. Elles sont considérées comme étant seulement visible depuis l'intérieur d'un composant (nœud).

Évènements : Cet ensemble représente tous les évènements provenant de l'environnement qui peuvent avoir une action sur le composant. Un évènement agit sur au moins une variable d'état à travers une transition. Les évènements peuvent être ordonnés avec des priorités : si deux évènements sont possibles au même moment, seul celui qui a la plus haute priorité aura lieu. Si deux évènements ont des priorités identiques, ils sont tous les deux possibles et on obtient alors une transition stochastique.

Transitions : Une transition représente une arête de l'automate interne. Une transition a une garde (expression booléenne), qui donne une pré-condition pour que la transition puisse être activée. Une transition est associée à un évènement et cause des changements sur une ou plusieurs variables d'état.

Assertions : Une assertion est une expression booléenne décrivant des invariants sur des variables. Elles servent en particulier à établir des connexions entre les variables de flux d'entrée et celles de sortie. Lorsqu'on modélise un système, les transitions sont utilisées pour modifier les variables d'état, tandis que les assertions permettent de modifier les variables de flot.

Initialisations : Un nœud doit affecter une valeur initiale à ses variables d'état.

Sous-nœuds : Un nœud peut contenir d'autres nœuds, représentant ainsi des sous systèmes d'un composant. En déclarant un nœud, on définit en vérité une classe d'objets qui peut être instanciée plusieurs fois (c'est à dire qu'on définit un objet abstrait, un patron, à partir duquel plusieurs objets "réels" peuvent être créés). Lorsqu'on déclare une instance d'un nœud à l'intérieur d'un autre nœud, on autorise le nœud père à accéder aux variables de flots et aux évènements du nœud fils. Les variables de flots des nœuds fils peuvent ainsi être connectées entre elles, ou être utilisées dans les assertions du nœud père. Le nœud père peut aussi synchroniser ou établir des

priorités entre les évènements de ses sous-nœuds. Hormis pour l'initialisation, le nœud père ne peut accéder aux variables d'état de ses sous-nœuds. Un nœud fils ne peut accéder à aucune des variables d'un nœud père.

Synchronisations : Des évènements peuvent être synchronisés pour avoir lieu en même temps. Ce mécanisme est déclaré au travers de vecteurs de synchronisations (des vecteurs contenant des noms d'évènements). Cette synchronisation est dite forte, puisqu'il suffit qu'un des évènements du vecteur de synchronisation ne puisse pas avoir lieu (c'est à dire qu'aucune de ses transitions n'a toutes ses pré-conditions remplies) pour qu'aucun des évènements concernés n'aient lieu. Le langage Altarica possède un mécanisme pour contourner cette contrainte, en autorisant de marquer certains évènements avec un '?' pour déclarer une **synchronisation faible** : des évènements marqués ainsi participeront à la synchronisation si possible, mais ne sont pas obligatoires. Il est possible d'être encore plus précis en déclarant une contrainte sur le nombre minimal ou maximal d'évènements devant participer à la synchronisation.

ANNEXE G

CHAÎNE DE DONNÉES RNP-AR

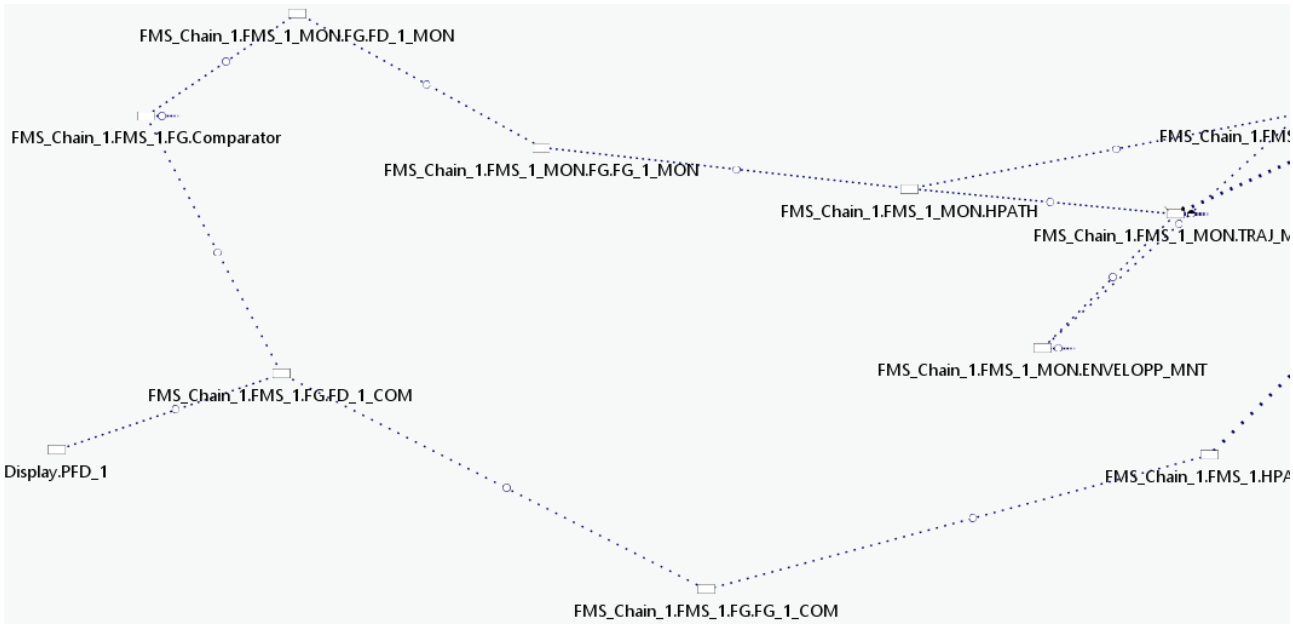


FIGURE 42 – Extrait du modèle de la chaîne de données RNP-AR

TABLE DES FIGURES

Chapitre V

Chapitre VI

Chapitre VII

Chapitre VIII

Chapitre B

Chapitre D

Chapitre G

LISTE DES TABLEAUX

LISTE DES ALGORITHMES

LISTE DES DÉFINITIONS

BIBLIOGRAPHIE

[900] ISO 9000. Iso 9000 - quality management. `http://www.iso.org/iso/home/standards/management-standards/iso_9000.htm`. Accessed : 2015-11-01. (Cité page 220.)

[Abr96] J.-R. Abrial. *The B-book : Assigning Programs to Meanings*. Cambridge University Press, New York, NY, USA, 1996. (Cité page 37.)

[AD92] R. Amalberti and F. Deblon. Cognitive modelling of fighter aircraft process control : a step towards an intelligent on-board assistance system. *International Journal of Man-Machine Studies*, 36(5) :639–671, 1992. (Cité page 214.)

[Ade11] R. Adeline. *Méthodes pour la validation de modèles formels pour la sûreté de fonctionnement et extension aux problèmes multi-physiques*. PhD thesis, Toulouse, ISAE, 2011. (Cité pages 10, 181, 182, 205 et 232.)

[ALRL04] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1) :11–33, 2004. (Cité pages 10 et 129.)

[Alt99] E. Altman. *Constrained Markov Decision Processes*. Chapman and Hall/CRC, 1999. (Cité pages 4, 39 et 142.)

[AOK14] A. Altay, O. Ozkan, and G. Kayakutlu. Prediction of aircraft failure times using artificial neural networks and genetic algorithms. *Journal of Aircraft*, 51(1) :47–53, 2014. (Cité page 198.)

[APGR99] A. Arnold, G. Point, A. Griffault, and A. Rauzy. The altarica formalism for describing concurrent systems. *Fundam. Inf.*, 40(2-3) :109–124, November 1999. (Cité pages 5, 36 et 127.)

[Atl12] M. Atli. *Contributions à la synthèse de commande des systèmes à évènements discrets : nouvelle modélisation des états interdits et application à un atelier flexible*. PhD thesis, Université de Lorraine, 2012. (Cité page 13.)

[ATS] ATSB. Data entry error and tailstrike involving boeing 737-838, vh-vzr. `http://atsb.gov.au/media/5727742/ao-2014-162_final.pdf`. Accessed : 2015-11-18. (Cité page 200.)

[Baz15] M. Bazargan. An optimization approach to aircraft dispatching strategy with maintenance cost–a case study. *Journal of Air Transport Management*, 42 :10–14, 2015. (Cité page 199.)

[BBC+04] P. Bieber, C. Bougnol, C. Castel, J.P. Heckmann, C. Kehren, S. Metge, and C. Seguin. Safety assessment with altarica. In *IFIP Congress Topical Sessions*, page 505–510, Toulouse, France, 2004. Springer. (Cité page 37.)

[BBG96] F. Bacchus, C. Boutilier, and A. Grove. Rewarding behaviors. In *The Thirteenth National Conference on Artificial Intelligence (AAAI)*, 1996. (Cité page 40.)

[BBG97] F. Bacchus, C. Boutilier, and A. Grove. Structured solution methods for non-Markovian decision processes. In *The Fourteenth National Conference on Artificial Intelligence (AAAI)*, 1997. (Cité page 41.)

[BEA] BEA. Preliminary report, airbus a320-211. `http://www.bea.aero/docspa/2015/d-px150324.en/pdf/d-px150324.en.pdf`. Accessed : 2015-11-01. (Cité page 216.)

[Bea11] E. Beaudry. *Planification D'Actions Concurrentes Sous Contraintes Et Incertitude*. PhD thesis, Universite de Sherbrooke (Canada), 2011. AAINR75069. (Cité page 32.)

[Bel86] V. Belton. A comparison of the analytic hierarchy process and a simple multi-attribute value function. *European Journal of Operational Research*, 26(1) :7–21, 1986. (Cité page 111.)

[Ber95] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995. (Cité pages 4, 31, 79, 83, 91 et 92.)

[BETT98] G. Di Battista, P. Eades, R. Tamassia, and I. Tollis. *Graph drawing : algorithms for the visualization of graphs*. Prentice Hall PTR, 1998. (Cité page 177.)

[BF97] A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial intelligence*, 90(1) :281–300, 1997. (Cité pages 29 et 67.)

[BFKK08] T. Brázdil, V. Forejt, J. Kretínský, and A. Kucera. The satisfiability problem for probabilistic ctl. In *Proceedings of the 2008 23rd Annual IEEE Symposium on Logic in Computer Science*, LICS '08, pages 391–402, Washington, DC, USA, 2008. IEEE Computer Society. (Cité page 36.)

[BGL+04] C. Baier, M. Größer, M. Leucker, B. Bollig, and F. Ciesinski. Controller synthesis for probabilistic systems. In *IFIP TCS*, 2004. (Cité page 41.)

[BHHK03] C. Baier, B. Haverkort, H. Hermanns, and J. Katoen. Model-checking algorithms for continuous-time markov chains. *Software Engineering, IEEE Transactions on*, 29(6) :524–541, June 2003. (Cité page 35.)

[BKDD04] C.S. Byington, P.W. Kalgren, B.K. Dunkin, and B.P. Donovan. Advanced diagnostic/prognostic reasoning and evidence transformation techniques for improved avionics maintenance. In *Aerospace Conference, 2004. Proceedings. 2004 IEEE*, volume 5, pages –3434 Vol.5, March 2004. (Cité page 53.)

[BKH99] C. Baier, J.-P. Katoen, and H. Hermanns. Approximative symbolic model checking of continuous-time markov chains. In JosC.M. Baeten and Sjouke Mauw, editors, *CONCUR'99 Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 146–161. Springer Berlin Heidelberg, 1999. (Cité pages 35 et 70.)

[Boa] US National Transportation Safety Board. Loss of thrust in both engines after encountering a flock of birds and subsequent ditching on the hudson river, us airways flight 1549. `http://www.ntsb.gov/investigations/AccidentReports/Reports/AAR1003.pdf`. Accessed : 2015-11-01. (Cité page 216.)

[Bon01] H. Bonet, B.and Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1) :5–33, 2001. (Cité pages 152 et 153.)

[BP98] D. Bertsimas and S.S. Patterson. The air traffic flow management problem with enroute capacities. *Operations research*, 46(3) :406–422, 1998. (Cité page 222.)

[BP00] D. Bertsimas and S. S. Patterson. The traffic flow management rerouting problem in air traffic control : A dynamic network flow approach. *Transportation Science*, 34(3) :239–255, 2000. (Cité page 222.)

[BPW+12] C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1) :1–43, March 2012. (Cité page 31.)

[BR85] F. J. Beutler and K. W. Ross. Optimal policies for controlled markov chains with a constraint. *Journal of Mathematical Analysis and Applications*, 112(1) :236–252, 11 1985. (Cité page 39.)

[BRB91] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a bdd package. In *Proceedings of the 27th ACM/IEEE design automation conference*, pages 40–45. ACM, 1991. (Cité page 37.)

[CAW88] A.K. Caglayan, S.M. Allen, and K. Wehmuller. Evaluation of a second generation reconfiguration strategy for aircraft flight control systems subjected to actuator failure/surface damage. In *Aerospace and Electronics Conference, 1988. NAECON 1988., Proceedings of the IEEE 1988 National*, pages 520–529. IEEE, 1988. (Cité page 172.)

[CCG+02] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2 : An opensource tool for symbolic model checking. In *Computer Aided Verification*, pages 359–364. Springer, 2002. (Cité pages 37 et 182.)

[CdA07] C.A.V. Cavalcante and A.T. de Almeida. A multi-criteria decision-aiding model using promethee iii for preventive maintenance planning under uncertain conditions. *Journal of Quality in Maintenance Engineering*, 13(4) :385–397, 2007. (Cité page 111.)

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2) :244–263, April 1986. (Cité pages 4, 34 et 74.)

[CF07] R.C. Chen and E.A. Feinberg. Non-randomized policies for constrained markov decision processes. *Mathematical Methods of Operations Research*, 66(1) :165–179, 2007. (Cité page 149.)

[Cha08] F. Chauvel. *Méthodes et outils pour la conception de systèmes logiciels auto-adaptatifs*. PhD thesis, Université de Bretagne Sud, 2008. (Cité page 11.)

[Che80] B.F. Chellas. *Modal Logic : An Introduction*. Cambridge University Press, 1980. (Cité pages 20 et 39.)

[CL00] Y. Crozet and F. Lawson. La commercialisation des créneaux aéroportuaires : une fausse bonne idée ? *Transports*, (399) :26–35, 2000. (Cité page 222.)

[CMF96] C.C. Chou, D. Madhavan, and K. Funk. Studies of cockpit task management errors. *The International Journal of Aviation Psychology*, 6(4) :307–320, 1996. (Cité page 214.)

[CML14] M. Chen, S. Mao, and Y. Liu. Big data : A survey. *Mobile Networks and Applications*, 19(2) :171–209, 2014. (Cité page 198.)

[CO] COIN-OR. Clp solver. `http://www.coin-or.org/`. Accessed : 2015-11-08. (Cité page 66.)

[CR00] A. Cimatti and M. Roveri. Conformant planning via model checking. In *Recent Advances in AI Planning*, pages 21–34. Springer, 2000. (Cité pages 66, 141 et 142.)

[CTA04] A.J. Cook, G. Tanner, and S. Anderson. Evaluating the true cost to airlines of one minute of airborne or ground delay : final report. *Project Report*, 2004. (Cité pages 51 et 120.)

[CY90] Costas Courcoubetis and Mihalis Yannakakis. Markov decision processes and regular events. In M.S. Paterson, editor, *Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 336–349. Springer Berlin Heidelberg, 1990. (Cité page 40.)

[DGV15] G. De Giacomo and M.Y. Vardi. Synthesis for ltl and ldl on finite traces. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 1558–1564. AAAI Press, 2015. (Cité page 198.)

[Die00] S. Diebolt. Le petit lexique des termes de la complexité, 2000. (Cité page 10.)

[Dij] E. Dijkstra. Shunting-yard algorithm. `https://en.wikipedia.org/wiki/Shunting-yard_algorithm`. Accessed : 2015-11-08. (Cité page 179.)

[Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1) :269–271, 1959. (Cité page 29.)

[DL95] T. Dean and S.H. Lin. Decomposition techniques for planning in stochastic domains. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1121–1127, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. (Cité page 32.)

[DM10] E. Delage and S. Mannor. Percentile optimization for markov decision processes with parameter uncertainty. *Operations research*, 58(1) :203–213, 2010. (Cité page 121.)

[DS10] T. Degris and O. Sigaud. Factored markov decision processes. *Markov Decision Processes in Artificial Intelligence*, pages 99–126, 2010. (Cité page 119.)

[Dul15] D.A. Dulo. Unmanned aircraft : The rising risk of hostile takeover [leading edge]. *Technology and Society Magazine, IEEE*, 34(3) :17–19, 2015. (Cité page 200.)

[EASa] EASA. Airworthiness approval and operational criteria for rnp approach. `http://easa.europa.eu/system/files/dfu/agency-measures-docs-agency-decisions-2009-2009-019-R-Annex-III---AMC-20-27.pdf`. Accessed : 2015-11-08. (Cité page 173.)

[EASb] EASA. Master minimum equipment list. `http://easa.europa.eu/document-library/master-minimum-equipment-lists`. Accessed : 2015-11-08. (Cité pages 9, 51 et 107.)

[EFJ+98] M.R. Endsley, T.C. Farley, W.M. Jones, A.H. Midkiff, and R.J. Hansman. Situation awareness information requirements for commercial airline pilots. Technical report, International Center for Air Transportation, 1998. (Cité pages 140, 214, 215 et 218.)

[EGK+02] J. Ellson, E. Gansner, L. Koutsofios, S.C. North, and G. Woodhull. Graphviz-open source graph drawing tools. In *Graph Drawing*, pages 483–484. Springer, 2002. (Cité page 177.)

[EKVY07] K. Etessami, M. Kwiatkowska, M.Y. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. In *Proc. of the 13th int. conf. on Tools and algorithms for the construction and analysis of systems*, page 5065, Berlin, Heidelberg, 2007. SpringerVerlag. (Cité pages 4, 41 et 142.)

[Eme90] E. A. Emerson. Handbook of theoretical computer science (vol. b). In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter Temporal and Modal Logic, pages 995–1072. MIT Press, Cambridge, MA, USA, 1990. (Cité page 34.)

[End95] M.R. Endsley. Toward a theory of situation awareness in dynamic systems. *Human Factors : The Journal of the Human Factors and Ergonomics Society*, 37(1) :32–64, 1995. (Cité pages 214 et 215.)

[Est07] J.A. Estefan. Survey of model-based systems engineering (mbse) methodologies. *Incose MBSE Focus Group*, 25 :8, 2007. (Cité page 176.)

[FAA] FAA. Rnp ar approval guidance. `https://www.faa.gov/about/office_org/headquarters_offices/avs/offices/afs/afs400/afs470/rnp/`. Accessed : 2015-11-08. (Cité page 171.)

[FGE05] J. Figueira, S. Greco, and M. Ehrgott. *Multiple criteria decision analysis : state of the art surveys*, volume 78. Springer Science & Business Media, 2005. (Cité pages 63 et 111.)

[FH10] J. Fürnkranz and E. Hüllermeier. *Preference learning*. Springer, 2010. (Cité page 67.)

[FKN+11] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Quantitative multi-objective verification for probabilistic systems. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems : Part of the Joint European Conferences on Theory and Practice of Software*, TACAS'11/ETAPS'11, pages 112–127, Berlin, Heidelberg, 2011. Springer-Verlag. (Cité page 42.)

[FKP12] V. Forejt, M. Kwiatkowska, and D. Parker. Pareto curves for probabilistic model checking. In *Proceedings of the 10th International Conference on Automated Technology for Verification and Analysis*, ATVA'12, pages 317–332, Berlin, Heidelberg, 2012. Springer-Verlag. (Cité page 42.)

[FLV06] P.H. Feiler, B.A. Lewis, and S. Vestal. The sae architecture analysis & design language (aadl) a standard for engineering performance critical systems. In *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, pages 1206–1211. IEEE, 2006. (Cité page 220.)

[FMS14] S. Friedenthal, A. Moore, and R. Steiner. *A practical guide to SysML : the systems modeling language*. Morgan Kaufmann, 2014. (Cité page 128.)

[FR91] T.M.J. Fruchterman and E.M. Reingold. Graph drawing by force-directed placement. *Softw., Pract. Exper.*, 21(11) :1129–1164, 1991. (Cité page 177.)

[FR12] E.A. Feinberg and U.G Rothblum. Splitting randomized stationary policies in total-reward markov decision processes, 2012. (Cité page 39.)

[Fun91] K. Funk. Cockpit task management : Preliminary definitions, normative theory, error taxonomy, and design recommendations. *The International Journal of Aviation Psychology*, 1(4) :271–285, 1991. (Cité page 214.)

[Gar] The Gardian. App fail on ipad grounds 'a few dozen' american airlines flights. `http://www.theguardian.com/technology/2015/apr/29/apple-ipad-fail-grounds-few-dozen-american-airline-flights`. Accessed : 2015-11-18. (Cité page 200.)

[GJS03] J. Gozdecki, A. Jajszczyk, and R. Stankiewicz. Quality of service terminology in ip networks. *Communications Magazine, IEEE*, 41(3) :153–159, 2003. (Cité page 140.)

[GL10] M. Grabisch and C. Labreuche. A decade of application of the choquet and sugeno integrals in multi-criteria decision aid. *Annals of Operations Research*, 175(1) :247–286, 2010. (Cité page 111.)

[GLD00] R. Givan, S. Leach, and T. Dean. Bounded-parameter markov decision processes. *Artificial Intelligence*, 122(1) :71–109, 2000. (Cité page 121.)

[GNT04] M. Ghallab, D. Nau, and P. Traverso. *Automated planning : theory & practice*. Elsevier, 2004. (Cité pages 29, 142 et 165.)

[Goo] Google. Material design guidelines. `https://www.google.fr/design/spec/material-design/introduction.html`. Accessed : 2015-11-08. (Cité page 176.)

[GPT04] C. Gretton, D. Price, and S. Thiébaux. Nmrdpp : Decision-theoretic planning with control knowledge. In *In Proceedings of the Probablistic Planning Track of IPC-04*, 2004. (Cité page 40.)

[Gro09] R.C. Gronback. *Eclipse modeling project : a domain-specific language (DSL) toolkit*. Pearson Education, 2009. (Cité page 178.)

[GT00] F. Giunchiglia and P. Traverso. Planning as model checking. In Susanne Biundo and Maria Fox, editors, *Recent Advances in AI Planning*, volume 1809 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg, 2000. (Cité page 30.)

[GV04] A. Griffault and A. Vincent. The mec 5 model-checker. In *Computer Aided Verification*, pages 488–491. Springer, 2004. (Cité page 182.)

[HBG05] P. Haslum, B. Bonet, and H. Geffner. New admissible heuristics for domain-independent planning. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*, AAAI'05, pages 1163–1168. AAAI Press, 2005. (Cité page 32.)

[HC13] C. Hajiyev and F. Caliskan. *Fault diagnosis and reconfiguration in flight control systems*, volume 2. Springer Science & Business Media, 2013. (Cité page 172.)

[HCH+02] B. Haverkort, L. Cloth, H. Hermanns, J. Katoen, and C. Baier. Model checking performability properties. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 103–112, 2002. (Cité page 35.)

[HCM+00] S.R. Haddock, J.N. Chueh, S.T. Merchant, A.H. Smith, and M. Yip. Policy based quality of service, August 15 2000. US Patent 6,104,700. (Cité page 140.)

[Hel10] A. Helfrick. *Principles of avionics*. Avionics Communications, 2010. (Cité page 10.)

[HJ94] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, pages 512–535, 1994. (Cité pages 4, 9, 20, 21, 22, 35, 70, 76, 89, 102, 147 et 157.)

[HK02] B. Hasselblatt and A. Katok. *Handbook of dynamical systems*, volume 1. Elsevier, 2002. (Cité page 11.)

[HMPK+98] M. Hauskrecht, N. Meuleau, L. Pack Kaelbling, T. Dean, and C. Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *In Proc. of Uncertainty in Artificial Intelligence (UAI*, pages 220–229, 1998. (Cité page 31.)

[HNR68] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2) :100–107, July 1968. (Cité page 29.)

[HPL04] J. Hoffmann, J. Porteous, and Sebastia L. Ordered landmarks in planning. *Journal Artificial Intelligence Research*, 22 :215–278, 2004. (Cité page 165.)

[HRM07] E. Huang, R. Ramamurthy, and L.F. McGinnis. System and simulation modeling using sysml. In *Proceedings of the 39th conference on Winter simulation : 40 years ! The best is yet to come*, pages 796–803. IEEE Press, 2007. (Cité page 128.)

[HZ01] E.A. Hansen and S. Zilberstein. Lao : a heuristic search algorithm that finds solutions with loops. *Artif. Intell.*, 129(1-2) :35–62, June 2001. (Cité pages 4, 31 et 190.)

[HZTM09] A. Heng, S. Zhang, A.C.C Tan, and J. Mathew. Rotating machinery prognostics : State of the art, challenges and opportunities. *Mechanical Systems and Signal Processing*, 23(3) :724 – 739, 2009. (Cité page 54.)

[ICA] ICAPS. International planning competition. `http://www.icaps-conference.org/index.php/Main/Competitions`. Accessed : 2015-11-18. (Cité pages 32 et 164.)

[JM+09] A Jorge, Sheila A McIlraith, et al. Planning with preferences. *AI Magazine*, 29(4) :25, 2009. (Cité pages 135 et 189.)

[KD94] R.J. Kelly and J.M. Davis. Required navigation performance (rnp) for precision approach and landing with gnss application. *Navigation*, 41(1) :1–30, 1994. (Cité page 171.)

[Keh05] C. Kehren. *Systems architectures formal safety patterns*. Theses, Ecole nationale superieure de l'aeronautique et de l'espace, December 2005. (Cité page 12.)

[Kid99] P. Kidwell. Journey to the moon : the history of the apollo guidance computer. *Annals of the History of Computing, IEEE*, 21(1) :78–79, 1999. (Cité page 202.)

[KMW12] A. Kolobov, M. Mausam, and D. S. Weld. A theory of goal-oriented mdps with dead ends. In *UAI*, pages 438–447, 2012. (Cité pages 31, 69, 70 et 92.)

[KMWG11] A. Kolobov, M. Mausam, D. Weld, and H. Geffner. Heuristic search for generalized stochastic shortest path mdps. *International Conference on Automated Planning and Scheduling*, 2011. (Cité pages 4, 31, 79 et 83.)

[KP88] G.E. Krasner and S.T. Pope. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming*, 1(3) :26–49, 1988. (Cité page 176.)

[KP13] M. Kwiatkowska and D. Parker. Automated verification and strategy synthesis for probabilistic systems. In D. Van Hung and M. Ogawa, editors, *Proc. 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13)*, volume 8172 of *LNCS*, pages 5–22. Springer, 2013. (Cité page 37.)

[Kun13] F. Kuntz. *Une approche basée modèle pour l'optimisation du monitoring de systèmes avioniques relativement à leurs performances de diagnostic*. PhD thesis, Informatique Bordeaux 1, 2013. Thèse de doctorat dirigée par Gaudan, Stéphanie Griffault, Alain et Muscholl, Anca Informatique Bordeaux 1 2013. (Cité pages 51 et 53.)

[KZH+11] J.P Katoen, I.S. Zapreev, E.M. Hahn, H. Hermanns, and D.N. Jansen. The ins and outs of the probabilistic model checker mrmc. *Perform. Eval.*, 68(2) :90–104, February 2011. (Cité page 37.)

[LLH05] C. Labreuche and F. Le Huédé. Myriad : a tool suite for mcda. In *EUSFLAT*, volume 5, pages 204–209, 2005. (Cité page 111.)

[LM67] E. B. Lee and L. Markus. Foundations of optimal control theory. Technical report, DTIC Document, 1967. (Cité page 11.)

[LN09] Y.G. Li and P. Nilkitsaranont. Gas turbine performance prognostic for condition-based maintenance. *Applied Energy*, 86(10) :2152 – 2161, 2009. (Cité page 54.)

[LPH15] B. Lacerda, D. Parker, and N. Hawes. Optimal policy generation for partially satisfiable co-safe LTL specifications. In *Proc. 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 1587–1593. IJCAI/AAAI, 2015. (Cité page 198.)

[Lue73] D.G. Luenberger. *Introduction to linear and nonlinear programming*, volume 28. Addison-Wesley Reading, MA, 1973. (Cité page 142.)

[Mar06] A.G. Marin. Airport management : taxi planning. *Annals of Operations Research*, 143(1) :191–202, 2006. (Cité page 218.)

[MBB+14] N. Meuleau, E. Benazera, R.I. Brafman, E.A. Hansen, and M. Mausam. A heuristic search approach to planning with continuous resources in stochastic domains. *CoRR*, abs/1401.3428, 2014. (Cité page 40.)

[Mey92] B. Meyer. Applying "design by contract". *Computer*, 25(10) :40–51, October 1992. (Cité pages 128 et 130.)

[Mon82] G.E. Monahan. State of the art—a survey of partially observable markov decision processes : theory, models, and algorithms. *Management Science*, 28(1) :1–16, 1982. (Cité pages 30, 31 et 65.)

[MRWO14] R. Mehta, S. Rice, S.R. Winter, and K. Oyman. Consumers' perceptions about autopilots and remote-controlled commercial aircraft. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 58, pages 1834–1838. SAGE Publications, 2014. (Cité page 199.)

[MT00] N. Medvidovic and R.N. Taylor. A classification and comparison framework for software architecture description languages. *Software Engineering, IEEE Transactions on*, 26(1) :70–93, 2000. (Cité page 37.)

[Mül02] J.P. Müller. Des systèmes autonomes aux systèmes multi-agents : Interaction, émergence et systèmes complexes. *Rapport présenté pour l'obtention de l'Habilitationa Diriger les Recherches en Informatique*, 8, 2002. (Cité page 11.)

[NAI+03] D.S. Nau, T.C. Au, O. Ilghami, U. Kuter, J.W. Murdock, D. Wu, and F. Yaman. Shop2 : An htn planning system. *J. Artif. Intell. Res.(JAIR)*, 20 :379–404, 2003. (Cité page 67.)

[NGT04] D. Nau, M. Ghallab, and P. Traverso. *Automated Planning : Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. (Cité page 29.)

[NHR99] A.Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations : Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999. (Cité page 121.)

[Nor98] J.R. Norris. *Markov chains*. Cambridge series in statistical and probabilistic mathematics. Cambridge University Press, 1998. (Cité page 17.)

[Odo] Odoxa. Sondage la voiture autonome. `http://www.syntec-numerique.fr/sites/default/files/related_docs/rdv_odoxa_sn_la_voiture_autonome_27mai2015.pdf`. Accessed : 2015-11-01. (Cité page 217.)

[oT] United States Department of Transportation. Employment in for-hire transportation and selected transportation-related industries. `http://www.rita.dot.gov/bts/sites/rita.dot.gov.bts/files/publications/national_transportation_statistics/index.html`. Accessed : 2015-11-01. (Cité page 216.)

[Par07] T. Parr. The definitive antlr reference : building domain-specific languages. *The definitive ANTLR reference*, 2007. (Cité page 178.)

[PBBB04] H.K. Preisler, D.R. Brillinger, R.E. Burgan, and J.W. Benoit. Probability based models for estimation of wildfire risk. *International Journal of Wildland Fire*, 13(2) :133–142, 2004. (Cité page 92.)

[PCCT13] C. Ponzoni Carvalho Chanel and F. Teichteil-Königsbuch. Properly acting under partial observability with action feasibility constraints. In H. Blockeel, K. Kersting, S. Nijssen, and F. Zelezný, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part I*, volume 8188 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2013. (Cité pages 68 et 141.)

[PCJ96] H.C. Purchase, R.F. Cohen, and M. James. Validating graph drawing aesthetics. In *Graph Drawing*, pages 435–446. Springer, 1996. (Cité page 177.)

[PFB02] L. Pallottino, E.M. Feron, and A. Bicchi. Conflict resolution problems for air traffic management systems solved with mixed integer programming. *Intelligent Transportation Systems, IEEE Transactions on*, 3(1) :3–11, 2002. (Cité page 222.)

[Piz13] S. Pizziol. *Prédiction des conflits dans des systèmes homme-machine*. PhD thesis, Institut Supérieur de l'Aéronautique et de l'Espace (ISAE), 2013. (Cité page 214.)

[PM00] A.B. Piunovskiy and X. Mao. Constrained markovian decision processes : the dynamic programming approach. *Operations Research Letters*, 27(3) :119 – 126, 2000. (Cité page 39.)

[PNAL15] M. Phillips, V. Narayanan, S. Aine, and M. Likhachev. Efficient search with an ensemble of heuristics. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence. AAAI Press (to appear)*, 2015. (Cité page 198.)

[Pnu77] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57, Oct 1977. (Cité pages 9, 14, 20 et 34.)

[Pol] Polarsys. Capella. `https://www.polarsys.org/capella/`. Accessed : 2015-11-08. (Cité page 15.)

[Pos] The Washington Post. When drones fall from the sky. `http://www.washingtonpost.com/sf/investigative/2014/06/20/when-drones-fall-from-the-sky/`. Accessed : 2015-11-01. (Cité page 216.)

[Pro14] T. Prosvirnova. *AltaRica 3.0 : a Model-Based approach for Safety Analyses*. Theses, Ecole Polytechnique, November 2014. (Cité pages 37 et 127.)

[PT87] C. Papadimitriou and J.N. Tsitsiklis. The complexity of markov decision processes. *Math. Oper. Res.*, 12(3) :441–450, August 1987. (Cité page 30.)

[Put94] M. L. Puterman. *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc., New York, NY, USA, 1st edition, 1994. (Cité pages 2, 9, 23, 26, 30, 70, 75, 79, 84, 85, 136, 142 et 147.)

[PVLI10] C. Pralet, G. Verfaillie, M. Lemaître, and G. Infantes. Constraint-based controller synthesis in non-deterministic and partially observable domains. In *ECAI*, pages 681–686, 2010. (Cité pages 30 et 66.)

[PZ14] L. Pineda and S. Zilberstein. Planning under uncertainty using reduced models : Revisiting determinization. *International Conference on Automated Planning and Scheduling*, 2014. (Cité page 198.)

[Ram92] G. Ramohalli. The honeywell on-board diagnostic and maintenance system for the boeing 777. In *Digital Avionics Systems Conference, 1992. Proceedings., IEEE/AIAA 11th*, pages 485–490, Oct 1992. (Cité pages 52 et 53.)

[Rau02] A. Rauzy. Modes Automata and their Compilation into Fault Trees. *Reliability Engineering and System Safety*, 78(1) :1–12, 2002. (Cité page 37.)

[RBF14] A. Rauzy and C. Bleriot-Fabre. Model-based safety assessment : Rational and trends. In *Mecatronics (MECATRONICS), 2014 10th France-Japan/8th Europe-Asia Congress on*, pages 1–10. IEEE, 2014. (Cité page 200.)

[RCPF02] A. M. Rodrı´guez-Chı´a, J. Puerto, and F. R. Fernández. A multicriteria competitive markov decision process. *Mathematical Methods of Operations Research*, 55(3) :359–369, 2002. (Cité page 31.)

[San10] S. Sanner. Relational dynamic influence diagram language (rddl) : Language description. *Unpublished ms. Australian National University*, 2010. (Cité page 32.)

[SB97] D.L. Scapin and J.M.C. Bastien. Ergonomic criteria for evaluating the ergonomic quality of interactive systems. *Behaviour & information technology*, 16(4-5) :220–231, 1997. (Cité page 176.)

[Sel07] B. Selic. A systematic approach to domain-specific language design using uml. In *Object and Component-Oriented Real-Time Distributed Computing, 2007. ISORC'07. 10th IEEE International Symposium on*, pages 2–9. IEEE, 2007. (Cité page 178.)

[Shn92] B. Shneiderman. *Designing the user interface : strategies for effective human-computer interaction*, volume 3. Addison-Wesley Reading, MA, 1992. (Cité page 176.)

[SJ13] A. Soderberg and R. Johansson. Safety contract based design of software components. In *Software Reliability Engineering Workshops (ISSREW), 2013 IEEE International Symposium on*, pages 365–370. IEEE, 2013. (Cité page 130.)

[SKTK14] J. Sprauel, A. Kolobov, and F. Teichteil-Königsbuch. Saturated path-constrained mdp : Planning under uncertainty and deterministic model-checking constraints. *AAAI Conference on Artificial Intelligence*, 2014. (Cité pages 6, 149, 157 et 166.)

[SLJ73] J.K. Satia and R.E. Lave Jr. Markovian decision processes with uncertain transition probabilities. *Operations Research*, 21(3) :728–740, 1973. (Cité page 121.)

[SM14] D. Serban and J. McBride. Voice activated cockpit management system for flight procedures and control of aircraft systems and flight management systems of single and multi-engine aircraft, July 29 2014. US Patent 8,793,139. (Cité page 199.)

[Spe] Specinnovations. Drawings do not equal models. `http://www.specinnovations.com/drawings-do-not-equal-models/`. Accessed : 2015-11-08. (Cité page 177.)

[SS02] B. Schölkopf and A.J. Smola. *Learning with kernels : Support vector machines, regularization, optimization, and beyond*. MIT press, 2002. (Cité page 67.)

[SS04] J.W. Smeltink and M.J. Soomer. An optimisation model for airport taxi scheduling. *Informs Journal on Computing*, 2004. (Cité page 218.)

[SSS14] J. Sprauel, C. Sannino, and C. Seguin. Techniques d'Aide à la Décision appliquées à la maintenance d'un avion de type Business Jet Decision Aiding Techniques applied to the maintenance of a Business Jet. In *Congrès Lambda Mu 19*, DIJON, France, October 2014. (Cité page 6.)

[Ste95] A. Stentz. The focussed d* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995. (Cité page 67.)

[Sys] Dassault Systems. Catia. `http://www.3ds.com/products-services/catia`. Accessed : 2015-11-08. (Cité page 15.)

[TA07] K. Tumer and A. Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 255. ACM, 2007. (Cité page 222.)

[Tar72] R. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1972. (Cité pages 29 et 36.)

[Tec] Esterel Technologies. Scade. `http://www.esterel-technologies.com/products/scade-suite/`. Accessed : 2015-11-08. (Cité page 15.)

[TGS+06] S. Thiébaux, C. Gretton, J. Slaney, D. Price, and F. Kabanza. Decision-theoretic planning with non-Markovian rewards. *Journal Artificial Intelligence Research*, 25(1) :17–74, January 2006. (Cité page 40.)

[TK12a] F. Teichteil-Konigsbuch. Path-constrained markov decision processes : bridging the gap between probabilistic model-checking and decision-theoretic planning. *The Twentieth European Conference on Artificial Intelligence (ECAI)*, 2012. (Cité pages 4, 9, 42, 68, 70, 71, 77, 86, 91, 93, 96, 99, 149 et 162.)

[TK12b] F. Teichteil-Königsbuch. Stochastic safest and shortest path problems. In J. Hoffmann and B. Selman, editors, *AAAI*. AAAI Press, 2012. (Cité page 41.)

[TKIS11] F. Teichteil-Konigsbuch, G. Infantes, and C. Seguin. Lazy forward-chaining methods for probabilistic model-checking. In *European Safety And Reliability Conference(ESREL)*, Troyes, France, 2011. CRC Press. (Cité pages 36, 186 et 192.)

[TKVI11] F. Teichteil-Königsbuch, V. Vidal, and G. Infantes. Extending classical planning heuristics to probabilistic planning with dead-ends. In W. Burgard and D. Roth, editors, *AAAI*. AAAI Press, 2011. (Cité pages 4, 31 et 160.)

[VD06] Y. Vanderperren and W. Dehaene. From uml/sysml to matlab/simulink : current state and future perspectives. In *Proceedings of the conference on Design, automation and test in Europe : Proceedings*, pages 93–93. European Design and Automation Association, 2006. (Cité page 128.)

[Vil88] A. Villemeur. *Sûreté de fonctionnement des systèmes industriels : fiabilité, facteurs humains, informatisation.* Collection de la Direction des études et recherches d'Électricité de France. Eyrolles, 1988. (Cité pages 10 et 49.)

[VR03] H.G. Visser and P.C. Roling. Optimal airport surface traffic planning using mixed integer linear programming. In *AIAAs 3rd Annual Aviation Technology, Integration, and Operations (ATIO) Forum*, 2003. (Cité page 218.)

[VV98] V. Naumovich Vapnik and V. Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998. (Cité page 67.)

[WC96] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *Selected Areas in Communications, IEEE Journal on*, 14(7) :1228–1234, 1996. (Cité page 140.)

[wik] wikipedia.org. Required navigation performance. `https://en.wikipedia.org/wiki/Required_navigation_performance`. Accessed : 2015-11-08. (Cité page 171.)

[www] www.airbus.com. Airbus a320 family. `http://www.airbus.com/aircraftfamilies/passengeraircraft/a320family/`. Accessed : 2015-11-01. (Cité page 216.)

[Wym93] A.W. Wymore. *Model-based systems engineering*, volume 3. CRC press, 1993. (Cité page 220.)

[YFS12] C. Yoo, R. Fitch, and S. Sukkarieh. Probabilistic temporal logic for motion planning with resource threshold constraints. In *Robotics : Science and Systems VIII, University of Sydney, Sydney, NSW, Australia, July 9-13, 2012*, 2012. (Cité page 36.)

[YMS03] H.L.S Younes, D.J Musliner, and R.G. Simmons. A framework for planning in continuous-time stochastic domains. In *ICAPS*, pages 195–204, 2003. (Cité pages 5 et 32.)

[YS04] H.L.S Younes and R.G Simmons. Policy generation for continuous-time stochastic domains with concurrency. In *ICAPS*, pages 325–334, 2004. (Cité pages 41, 69, 101, 160 et 162.)

[ZSZM10] V. Zerbe, M. Schulz, A. Zimmermann, and S. Marwedel. Model-based evaluation of avionics maintenance and logistics processes. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 398–402, Oct 2010. (Cité page 54.)